(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
15 June 2000 (15.06.2000)

PCT

(10) International Publication Number
WO 00/034852 A1

(51) International Patent Classification⁶: G06F 3/14

(21) International Application Number: PCT/US99/29368

(22) International Filing Date: 10 December 1999 (10.12.1999)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/209,162 10 December 1998 (10.12.1998) US

(71) Applicant: INTELLINET, INC. [US/US]; 325 Riverside Drive, New York, NY 10025 (US).

(72) Inventor: BAKER, Michelle; 325 Riverside Drive, New York, NY 10025 (US).

(74) Agent: GORDON, David, P.; 65 Woods End Road, Stamford, CT 06905 (US).

(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW.

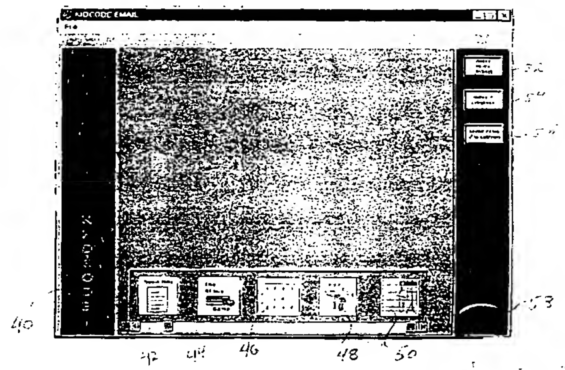
(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

(48) Date of publication of this corrected version:
29 August 2002

[Continued on next page]

(54) Title: AUTHORING E-MAIL WITH MODULAR COMPONENTS



(57) Abstract: Electronic mail software includes a main email component (40) and a number installable components (42, 44, 46, 48, 50). The installable components (42, 44, 46, 48, 50) include authoring/reading components for creating/reading different kinds of documents and mailbox components for listing different kinds of messages or for listing messages in different styles. The main email component (40) provides an underlying graphical user interface for functions directly associated with the storage and transfer of electronic mail messages, and also handles all data bundling and unbundling required to transform a message created by an authoring component into a MIME compliant message. The authoring/reading components (42, 44, 46, 48, 50) act like applications embedded within the email program and allow specific types of documents such as spreadsheets, graphics, databases, etc. to be created from within the email program and emailed directly. The authoring/reading components (42, 44, 46, 48, 50) also allow received documents to be read without the difficulties traditionally associated with attaching binary files to an email letter. The authoring components of the invention pass data to the main email component (40) which packages the data as a MIME compliant message. When the message is received, the main email component concatenates and decodes the MIME message and sends the data to the authoring/reading component associated with the MIME type.

WO 00/034852 A1

(15) Information about Correction:
see PCT Gazette No. 35/2002 of 29 August 2002, Section II

AUTHORING E-MAIL WITH A MODULAR COMPONENTS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to an electronic mail program. More particularly, the invention relates to an electronic mail program having modular integral authoring/reading applications whereby documents created with the modular integral authoring/reading applications are seamlessly sent and received by the mail program.

2. State of the Art

In recent years electronic mail ("email") has become widely used in business, education, and in personal communications. One of the features of electronic mail which is most convenient, particularly in business and in education, is the ability to attach a binary computer file to an email message. This feature enables email correspondents to rapidly share word processing documents, database documents, spreadsheet documents, multimedia documents, or virtually any kind of binary file created by a computer. There are, however, some serious limitations and inconveniences associated with attaching a binary file to an email message.

The original Internet mail system as defined in 1982 with RFC (Request for Comments) 821 and 822 had a number of important limitations. In particular, the system was not designed to carry large quantities of arbitrary data in an email message. In fact, the 1982 SMTP (Simple Mail Transport Protocol) standard required that an email message consist of a single message containing only ASCII characters in lines of 1000 characters (blocks of 32k) or less. Moreover, some implementations of SMTP or other mail transport systems (such as UUCP) restricted message lengths to some allowed maximum number of bytes. Messages passing through a mail gateway using one of these implementations were likely to be truncated.

The ability to send large quantities of binary data through the Internet electronic mail system was made possible with the MIME (Multipurpose Internet Mail Extensions) standard for Internet messages. The original MIME standard was published as an Internet Request For Comments document (RFC 1341) and approved in June of 1992. (See Internet RFCs 2045,2046, and 2047 for the latest MIME standards documents.) The MIME standard describes how an email message should be formatted in order to be considered MIME compliant. MIME defines a set of message header fields and a set of message encoding standards that are designed to overcome the limitations of RFC 822 message formats and still be transportable through any of the numerous legacy mail transport systems in use on the Internet.

MIME message header fields extend those defined in RFC 822 and describe the content and encoding type of the email message. Encoding schemes allowed in the MIME standard include "quoted-printable", and "base64". In addition, three unencoded data types are allowed. These are labeled "8bit", "7bit", or "binary".

If the sender and the receiver of the email message with the attached binary file are using the same brand and version of email program and both programs are configured in substantially the same way, the receiver's email program should automatically apply the appropriate decoding to the attached binary file and produce a file which is identical to the file which was attached to the email by the sender. However, if the sender and receiver are using different email programs, the recipient may receive a file which must be decoded by the recipient using a separate decoding program. Worse yet, if there is a failure of the receiving email program to properly deal with the MIME protocol, it is possible that the receiver will receive multiple files (each being ≤ 32k) which must first be concatenated and then decoded.

Even after the file is properly received and decoded, it is often difficult for the receiver of the file to open the file. The receiver of the file might expect that "clicking" on the file icon will open the file. However, clicking on the file icon will often not open the file. It may result in an error message like "application not found" or, worse, it may result in the file being opened by an inappropriate application thereby displaying "gibberish". The receiver of the file must have a program capable of reading (opening) the file. For example, if one attaches a spreadsheet file to an email message, the receiver of the file must have a spreadsheet program in order to open the file. Technically, it is not necessary that the receiver of the file have the same brand program as that which created the file. However, opening a file with a program which did not create it, though possible, can be very inconvenient. The receiver of the file must know what kind of file is attached to the email message, must know what program on their computer is capable of reading that type of file, must launch the program, must open the file from within the program, and wait while the program translates the file.

The limitations of Internet electronic mail can become even more frustrating if the sender and recipient are not using the same operating system (OS). Some mail attachment encoding schemes (and file compression schemes) are OS-dependent and it is possible that an email recipient could receive a file which is impossible to decode (or decompress).

These limitations in electronic mail have discouraged many people, particularly non-sophisticated computer users, from attaching files to electronic mail messages. In fact, for some novice users, the task of launching one application to create a document, saving the document, launching a separate email application to create an email message, and then locating the saved document for attachment to an email message is daunting enough to discourage them. In addition, novice users often complain that after "downloading" a file attached to an email message they cannot find the file on their hard disk.

SUMMARY OF THE INVENTION

It is therefore an object of the invention to provide an electronic mail program which includes integrated authoring software whereby a document may be created and sent by email in a seamless manner.

It is also an object of the invention to provide an electronic mail program which includes integrated authoring/reading software whereby a document may be received and opened in a seamless manner.

It is another object of the invention to provide an electronic mail program which includes modular integrated authoring software whereby different kinds of documents may be created and sent by email in a seamless manner.

It is still another object of the invention to provide an electronic mail program which includes modular integrated authoring/reading software whereby different kinds of documents may be received and opened in a seamless manner.

It is another object of the invention to provide an electronic mail program which includes modular integrated authoring/reading software whereby the authoring/reading software and the email software present an interface which suggests that a single application is operating.

It is another object of the invention to provide an electronic mail program which includes modular integrated mailbox handling software whereby messages of different types are displayed in different ways in a mailbox listing.

It is still another object of the invention to provide an electronic mail program which includes modular integrated authoring/reading software wherein the functionality of the authoring/reading software is controlled by the "role" of the user when participating in an exchange of messages.

In accord with these objects which will be discussed in detail below, the electronic mail software of the present invention includes a main email component and a number of installable components which communicate bidirectionally with the email component. The installable components include authoring/reading components as well as at least one mailbox browser/editor component. The main email component provides an underlying graphical user interface (GUI) for functions directly associated with the storage and transfer of electronic mail messages. In particular, the main email component provides menu items which allow the user to SEND, READ, REPLY, FORWARD, DELETE, SAVE, PRINT, for example. The main email program also handles all data bundling and unbundling that may be required to transform a message created by an authoring component into a fully MIME compliant message. In addition, the main email component includes "hooks" (an application programming interface or API) for the attachment of the installable components. The authoring/reading components each provide functionality which is particular to the type of document the component is designed to create/display. For example, a text document authoring component includes word processing functionality such as font selection, margin setting, etc. A painting/drawing authoring component includes tools for line drawing, polygon creation, paint brush, paint can, eraser, etc. A spreadsheet authoring component displays a grid and includes formula creation tools as well as formatting tools. A database authoring tool includes tools for creating fields and records, for sorting and searching, for generating reports, etc. A photo editor authoring component includes various imaging editing tools including cropping tools, dodging and burning tools, filters, etc. A presentation authoring component includes tools for creating slides and slide shows. The authoring components act like applications embedded within the email program and allow specific types of documents such as spreadsheets, graphics, databases, etc. to be created from within the email program and emailed directly. In addition, the authoring components allow received spreadsheets, graphics, databases, etc. to be read by the email program without the difficulties traditionally associated with attaching binary files to an email letter. According to the invention, in lieu of authoring components which allow both authoring and reading, separate components may be provided for authoring and reading, or components for reading only may be provided in addition to components which permit authoring as well as reading. The authoring/reading components interface with the main email component via designated "MIME types". The MIME data standard allows developers to define MIME types using the label "/application-x" in the data header. The authoring components of the invention pass data to the main email component which packages the data as a MIME compliant message with the label "/application-x" in the message header, where x identifies the authoring/reading component which created/can display the message. When the message is received, the main email component concatenates and decodes the MIME message, reads the MIME type, sends the data to the component associated with the MIME type, and waits for a user event or a callback from the component. This bidirectional communication between the main email component and the authoring/reading components provides a totally seamless operation wherein the user may send and receive complex documents without any knowledge of attaching files, downloading, decoding, etc.

The mailbox browser/editor (mailbox handler) component is provided preferably as a separate component rather than as part of the main email component so that the software may be more easily customized and upgraded. The mailbox browser/editor component is used to display, edit, and browse mailboxes. Since the invention provides for email messages which contain different kinds of data, the features of the mailbox browser may depend on the type of messages being sent and received. For example, if a graphical authoring/reading component is installed, it may be desirable to provide a mailbox browser which shows a thumbnail of the selected graphic email message when a list of messages is displayed.

The software according to the invention provides a single seamless environment for authoring, reading, and emailing a variety of different types of documents. The user does not need to understand uploading, downloading, file types, file decoding, or any of the other esoteric requirements of attaching files to email. Further, the user does not need to know what kind of application must be launched in order to read a particular type of email message.

An exemplary embodiment of the invention is an email program for school children called KIDCODE®. The KIDCODE® program includes a main email component, a mailbox browser/editor component and several message authoring/reading components. The main email component and the mailbox browser/editor component provide the same functionality as described above. Additional KIDCODE® components include a text authoring tool, rebus game message handler components (encoding and decoding components) which allow children to create and respond to graphical rebus messages, several different game puzzle components, and a workbook which allows a teacher to send workbook problems to a student and allows the student to send the solved problems back to the teacher. According to one inventive aspect of the invention which is exemplified in the workbook and rebus components, an authoring/reading component may assign and track user "roles" by associating a role tag to each message. For example, in the rebus component, the user initiating the rebus exchange will be assigned the role of rebus encoder. The message created by this user will contain a tag identifying it as an "encoded message". When the message is opened by the recipient, the tools available in the rebus component will be different from those available if a message were being encoded. Similarly, the workbook component is preferably provided with a teacher role and a student role, each of which have different tools. Component roles may be selected by the users, assigned by the system administrator, or automatically by components when messages are created/read.

According to a presently preferred embodiment, the KIDCODE® client software is written in the MACROMEDIA DIRECTOR™ LINGO™ scripting language which is cross-platform and thus ideally suited for use in schools which typically have a combination of MAC/OS™ and WINDOWS™ computers. As implemented, the client software operates over a TCP/IP LAN which is the most common type of network used in schools today and is compatible with the Internet. According to a further implementation of the invention, KIDCODE® software permits messages to be sent via the Internet in MIME compliant format.

Additional objects and advantages of the invention will become apparent to those skilled in the art upon reference to the detailed description taken in conjunction with the provided figures.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a screen shot of the KIDCODE® client login screen;

Figure 1a is a simplified state diagram illustrating the entry from the login screen to the main email component of the KIDCODE® software;

Figure 2 is a screen shot of the KIDCODE® main email component screen showing a menu of the installed authoring/display components and the mailbox browser component;

Figure 2a is a simplified state diagram illustrating the entry from the main email component into the installed components;

Figure 3 is a screen shot of the KIDCODE® mailbox browser/editor component screen;

Figure 4 is a screen shot of the KIDCODE® text message authoring component screen;

Figure 5 is a screen shot of the KIDCODE® rebus authoring (encoding) component screen;

Figure 6 is a screen shot similar to Figure 5 illustrating a listbox of users on the network to whom mail may be sent;

Figure 7 is a screen shot similar to Figure 5 illustrating a rebus in the process of being coded by the user;

Figure 8 is a screen shot of the KIDCODE® rebus reading (decoding) component screen;

Figure 9 is a screen shot of the KIDCODE® workbook authoring component screen;

Figure 10 is a screen shot illustrating the main email component of a second embodiment of the invention;

Figure 11 is a screen shot illustrating a text authoring component in the second embodiment of the invention;

Figure 12 is a screen shot illustrating a painting/drawing authoring component in the second embodiment of the invention;

Figure 13 is a screen shot illustrating a spreadsheet authoring component in the second embodiment of the invention;

Figure 14 is a screen shot illustrating a database authoring component in the second embodiment of the invention;

Figure 15 is a screen shot illustrating a photo editor authoring component in the second embodiment of the invention;

Figure 16 is a screen shot illustrating a slide show authoring component in the second embodiment of the invention; and

Figure 17 is a screen shot illustrating a display-only component in the second embodiment of the invention.

BRIEF DESCRIPTION OF THE APPENDICES

Appendix A is the LINGO™ script implementation of the KIDCODE® main email component;

Appendix B is a description of the Application Programming Interface for the KIDCODE® main email component which enables installable components to operate with the main email component.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

As mentioned above, a presently implemented embodiment of the invention is realized utilizing the MACROMEDIA DIRECTOR LINGO™ scripting language. The DIRECTOR™ application was originally intended to be used for authoring interactive multimedia presentations. Therefore, many of the terms used in the LINGO language refer to multimedia objects such as movies, cast members, frames, and sprites. However, it has been found that the LINGO™ language can be used to author many different types of programs including programs which were not traditionally thought of as being multimedia presentation programs. The following description, therefore, of the presently implemented embodiment will be best understood by those familiar with the MACROMEDIA DIRECTOR LINGO™ scripting language. However, those skilled in the art will understand from the functional description which follows that the invention could be implemented in other languages such as C or C++, JAVA™, etc.

Referring now to Figures 1 and 1a, and with reference to Appendix A, the first screen 10 presented by the KIDCODE® program is preferably a login screen where the user enters his or her name and password. According to the presently preferred embodiment, the login name field 12 is implemented as a popup menu (or pull down list box) and the password field 14 is a standard text entry field. See, for example, lines 172-190 of Appendix A. Clicking on the login name field will make a list of names appear and allow the user to highlight one of the names using the mouse. After the user has selected a name and typed in a password, the Okay button 16 must be clicked, or the Return or Enter key may be pressed. See Appendix A, lines 796-846 and lines 879-899. At this screen 10, the only option available under the FILE menu is QUIT. According to the presently preferred embodiment, the usernames and passwords are stored in associative (two property) lists so that a password may be associated with a username and a username may be associated with a password. When the okay button is clicked, the software checks the validity of the username and password. The checking of the username and password is illustrated in the state diagram in Figure 1a. Starting at 20 in Figure 1a, if the user selects a username and clicks the okay button, the password field is checked at 22. If no password was entered, a popup message is displayed at 24 indicating to the user that a password must be entered and the system returns to start at 20 waiting for the user to click the okay button. If the user types in a password and clicks the okay button, the username field is checked at 26. If no username was selected, a popup message is displayed at 24 indicating that a username must be selected and the system returns to start at 20 waiting for the user to click the okay button. If the user types in a password and selects a username, it is determined at 28 or 30 whether the username and password match, and if they do, the software enters to the main email component at 32. If the username and password do not match, a popup message is displayed at 34 indicating that the password entered is invalid for the username selected and the system returns to start at 20 waiting for the user to click the okay button. If the username and password are that of the system administrator, a special administration display will be shown in which usernames and passwords may be added/deleted to/from the system. See Appendix A lines 858-875, 900-1016, and 1123-1140.

Turning now to Figures 2 and 2a, once the user has selected a username and entered the correct password, the program displays the screen 40 shown in Figure 2 (Appendix A lines 851-855). This is the screen of the main email component with no other component selected. The screen 40 includes a scrollable collection of icons 42, 44, 46, 48, 50 and includes buttons 52, 54, 56 for mailbox access and button 58 to quit the program. The icons 42, 44, 46, 48, and 50 represent the installed authoring/reading components. As shown in Figure 2, the icons represent a text component 42, a rebus component 44, a "text in grid" component 46, a puzzle component 48, and a "connect the dots" component 50. As illustrated in Figure 2a, starting at 60, if the user clicks on the text icon, the program will, at 62, open the text authoring component with an empty message (Appendix A lines 1422-1438 and 1025-1054). Similarly, if the user clicks on the rebus icon, the program will, at 64, open the rebus authoring component with an empty message (Appendix A lines 1442-1458 and 1025-1054). The main email component will also, when an authoring component is opened, make the print, trash, and send buttons visible as these functions are served by the main email component as illustrated in Figure 2a (Appendix A lines 489-526 and 1456). Since the presently implemented example does not yet have all components complete, the launching of other components is shown generically at 66 in Figure 2a. When a component is launched, the main email program suspends execution at 68 awaiting any "call back" from the launched component. The API described in Appendix B includes a set of "call back" functions that can be used by an authoring or mailbox component to communicate with the main email component. See Appendix B pages 4-7 and Appendix A lines 557-744. Further, the main email component serves the email functions of inbox, outbox, and filing cabinet for messages that have not been sent. These functions are illustrated in Figure 2 as buttons 52, 54, 56 and in Figure 2a as routines 72, 74, 76.

For example, as shown in Figure 3, when the user enters the inbox, a window 78 and a button bar 80 are displayed. The mailbox component with the appropriate set of messages is launched as shown in Appendix A lines 1533-1558 and 216-247). The window 78 displays a list of new email and the button bar 80 displays buttons for functions common to all components, i.e. buttons for reply 82, send 84, print 86, and trash 88. It will be understood that depending on whether the user is in an authoring or reading mode, either the reply button 82 or the send button 84 will be "grayed out" indicating that that option is not available. As shown in Figure 3, for example, the send button 84 is not available when looking at the mail inbox. It will also be appreciated that the buttons and icons from the previous screen (Figure 2) are no longer visible. As stated in Figure 2a, the user returns to the screen of Figure 2 when the window 78 is closed.

The presently implemented text authoring/reading component is illustrated in the authoring mode in Figure 4. The window 90 is similar to any email message authoring tool and includes fields for "to:" 92, "from:" 94, "date" 96, "subject" 98, and "message" 100. The "from:" field 94 and "date" field 96 are hidden in Figure 4 behind the scrollable list box 102. After the addressee is chosen from the list box 102, the box disappears and reveals the "from" and "date" fields. According to the presently preferred embodiment, the list box 102 lists the names of all of the users registered in the local email system. This is handled by a call to the API as indicated at Appendix A lines 726-731. Thus, this embodiment prevents users from sending a document to a recipient who is not registered with the system administrator. It also allows users of the system to address messages without typing the recipient's name. According to another embodiment of the invention, shown and described below with reference to Figure 10, users are permitted to send email to any internet address and a list box is optionally used to display an address book. As seen in Figure 3, the reply button 82 in the button bar 80 is grayed out because that function is not available when authoring a text message. This is accomplished in Appendix A at lines 1422-1438.

Figures 5-8 illustrate the presently implemented rebus component. The rebus component presents a window 104 which includes a "to:" field 106, a message area 108, a "T" button 110 for selecting a template sentence, an "S" button 112 for hiding/displaying symbols, a "-" button 114 for hiding/displaying guesses, and a scrollable area 116 to the right of the message area which displays sets of symbols to be used in coding a rebus. In addition, the rebus component displays several pull down menus which are particular to it. These include the template choices menu 118 and the symbol choices menu 120.

According to the presently implemented embodiment, the author of a rebus begins by selecting a template sentence from a selection of sentences which are capable of being expressed as a rebus using the symbol sets provided. The template selection may be made via the T button

110 or the pull down menu 118. When a template sentence is selected, a suggested set of symbols is displayed in the field 116. Different symbols may be viewed by selecting a symbol set from the Symbol Choices menu 120. As illustrated in Figure 8, symbols are grouped according to the kinds of words they symbolize such as "action symbols", "object symbols", "quantity symbols", etc. As with other components of the KIDCODE® program, and as shown in Figure 6, the "to:" field 106 presents a drop down or pop up list box through which the email is addressed by selecting a registered user.

The author of the rebus codes the template sentence by dragging symbols from the scrollable field 116 to the message area 108. This is best illustrated by Figure 7. Symbols, when placed on a coded word in the template sentence, will snap into place when they are dragged into the area 108. According to the invention, not every word in the template sentence is designed to be coded with a symbol. According to the presently preferred embodiment, words which are to be coded appear in red text. For example, as shown in Figure 7, two symbols have been dragged into the message area and have snapped over the now hidden words "threw" and "window". The words "ball and "through" are also red text and can be coded with a proper symbol. The author can hide the symbols and display the words in the sentence which are covered by symbols by clicking on the "S" button 112. However, when the recipient receives the rebus, clicking on the button "S" will not reveal the words beneath the symbols, but will only make the symbols disappear. This is an example of how user "roles" alter the tools available in a component. When the author finishes coding the rebus, he or she clicks on the send button 84. The main email component then automatically encodes the rebus as a MIME attachment to Internet mail and sends the mail to the recipient's mailbox.

Turning now to Figure 8, when the recipient of the rebus opens the email message containing a rebus, the KIDCODE® main email component automatically decodes the MIME attachment, determines that it is a rebus, and opens it in the rebus reading component. The message appears with empty text boxes (e.g. 122, 124, 126) beneath the graphic symbols. The recipient of the message must solve the rebus by typing in the text boxes the words which he/she believes are represented by the graphic symbols. As mentioned above, the "-" button 114 is for hiding/displaying the guesses typed in the boxes. When the recipient has typed in words for all the graphic symbols, he/she clicks on the reply button 82 to send the solution back to the author. Figure 8 shows the screen after the button 82 has been clicked. Thus it is grayed out to prevent the same message from being sent twice. The palette 116 is available to the decoder for browsing only. The features which allow symbols to be placed on the message are disabled for the decoder.

One of the authoring/reading components of the invention is a workbook message handler, an example of which is illustrated in Figure 9. The screen shot shown in Figure 9 illustrates the "student role" of a workbook message handler. The window 130 of the student role workbook message handler preferably includes "to:" and "from:" fields 132, 134 which are filled in by the teacher before the message is sent to the student, as well as six fields 136, 138, 140, 142, 144, 146 which must be filled in by the student before the message is returned to the teacher. As shown in Figure 9, the window 130 also includes a title 148, a date 150 and various instructions 152. Those skilled in the art will appreciate that the date 150 may be automatically entered when the message is sent to the student. The fields 136, 138, 140, 142, 144, 146, the title 148, and the instructions 152 may be manually entered by the teacher or may be selected as part of a template. In other words, the workbook message handler component may be a complex tool which allows teachers to author an infinite number of "problem messages" to students or it may be a modular set of pre-written problems or templates for problems. The workbook message handler component preferably includes many pre-written problems. Additional pre-written problems will be available through additional modular components. One important feature of the workbook message handler components is that they identify user status and automatically present the proper "role" of either teacher or student.

The KIDCODE® program described above is designed to be easy to use by very young children. Figures 10-17 illustrate another embodiment of the invention which is designed for a more sophisticated user, an older child, or an adult. The interface is more complicated, but offers the user more features. Referring now to Figure 10, the interface of the second embodiment of the invention includes a menubar 200 which lists some standard menus like FILE, EDIT, SETUP, WINDOW, and HELP. The menubar 200 may also include a clock display 202 which is typically supplied by the operating system and a blinking icon 204 which is typically supplied by TCP/IP connection software such as PPP dialup software, to indicate that the computer is connected to the Internet. The menus MAIL and FOLDER in the menubar 200 are particular to the emailing program and the scroll icon 206 is provided by scripting software, typically part of the operating system. Under the MAIL menu, one will find commands such as NEW for creating new mail, REPLY for replying to a mail message being read, FORWARD, etc. Under the FOLDER menu, one will find the names of user created filing cabinets (folders) where incoming mail can be saved. Under the SETUP menu, one will find commands for setting the necessary information to make a connection with the Internet, for storing the user's name and password, for scheduling automatic sending and receiving of mail, for performing automated tasks in response to mail (e.g. for automatically filing certain mail in certain folders, autoresponding to certain mail, etc.), etc. Under the WINDOW menu, the user will have the option of viewing INBOX, OUTBOX, FILING CABINET(s), CONNECTION STATUS, etc.

The HELP menu preferably provides a context sensitive alphabetical list of help items which are hot linked to html files.

Figure 10 illustrates the email program with a new outgoing message window 208 opened. The message window includes a standard text message field 210, a standard subject field 212, standard multiple recipient address fields 214, and a variety of buttons. The arrow buttons 216 allow the user to scroll among messages in the outbox. The outbox button 218 drops down a list of items in the outbox, from which items may be selected. The magnifying glass button 220 is used to search the user's address book. The "+" button 222 adds a recipient to the address field 214. The trash button 224 places the outgoing message in the trash and closes the window 208. The clock button 226 brings up a menu to schedule when the message will be sent. The rotating arrow button 227 causes the address fields 214 to disappear/reappear thereby expanding/contracting the size of the message field 210. The send button 228 sends the message to the outbox (if it is scheduled for delivery at another time or if the computer is not connected to the Internet) and sends the message otherwise. The button 230 labelled " -^-K" causes the computer to connect to the Internet. As shown in Figure 10, this button 230 is grayed out because, as indicated by the blinking telephone pole icon 204, the computer is already connected to the Internet. The "send via" button 232 allows the user to select from several usernames, email accounts, etc.

The outgoing message window 208 shown in Figure 10 allows the user to send standard Internet mail by typing a message in the window 210. However, according to the invention, the window 208 also includes buttons 234, 236, 238, 240, 242, and 244, each of which is linked to an installed authoring/reading component. As described above, the number and nature of the authoring/reading components is modularly changeable. In the example shown in Figure 10, six authoring/reading components are shown to be installed. In practice, more, fewer, and/or different components may be installed. The components shown and described herein are: a word processor authoring/reading component linked to the button 234, a painting/drawing authoring/reading component linked to the button 236, a spreadsheet authoring/reading component linked to the button 238, a database authoring/reading component linked to the button 240, an image editor authoring/reading component linked to the button 242, and a presentation authoring/reading component linked to the button 244.

Turning now to Figure 11, when the user clicks on the button 234, the word processor component is invoked and it causes a new menubar 250 and a ruler 252 to appear inside the message field 210 of the window 208. The word processor component allows sophisticated formatting of messages which would be impossible in a normal Internet email program. For example, margins can be set using the ruler 252; fonts can be changed using the FONT menu from the menubar 250; tables can be created and inserted using the TABLE menu from the menubar 250. In general, the menubar 250 provides much or all of the functionality of a full featured word processor program. Those skilled in the art will appreciate that the word processor interface shown in Figure 11 is similar to the interface of Microsoft® Word® 98. It will be noted that the menubar 250 provides a separate HELP menu in addition to the HELP menu provided on the menubar 200. It will be appreciated that the HELP menu could be omitted from the menubar 250 and the help files for the word processor component could be accessed from the main HELP menu on the menubar 200. It will also be noted that when the word processor component is invoked, the button 234 is grayed.

After a user creates a message with the word processor component, the addressing and mailing procedure is the same as sending an ordinary email. There is no need to save a file, encode it, or attach it to an email message. The main email component of the invention seamlessly performs all of the saving, encoding, and attaching without any of this being exposed to the user. More particularly, the authoring component and the main email component cooperate to save the authored document as a file on the user's disk. See Appendix B lines 229-238 and Appendix A lines 1293-1333 and 329-450. The main email component encodes the file in the MIME format with as many parts as necessary, and sends the MIME file(s) as Internet email message(s). When the message is received by a person using a copy of the email program of the invention, the receiver's main email component seamlessly concatenates the MIME parts, decodes the MIME file determines that it is a message created with the word processing component (Appendix A lines 690-694), invokes the word processing component (Appendix A lines 1019-1054), and opens the message with the word processing component (Appendix A lines 603-614). The receiver of the message does not have to download any file, find any attachment, execute any decoders, or launch any word processor to see the fully formatted document created by the sender.

Turning now to Figure 12, when the user clicks on the button 236, the painting/drawing component is invoked and it causes a new menubar 260 and a tool palette 262 to appear inside the message field 210 of the window 208. The painting/drawing component allows the author to create a painting (bitmap) graphic or a drawing (vectormap) graphic and send it to another user for viewing/editing. Those skilled in the art will appreciate that the menubar 260 and palette 262 shown in Figure 12 contain the menus and tool icons typically found in a full featured drawing/painting program. Those skilled in the art will appreciate that the painting/drawing component interface shown in Figure 12 is similar to the interface of Aldus® SuperPaint® 3.5. It will be noted that the menubar 260 provides a separate HELP menu in addition to the HELP menu provided on the menubar 200. It will be appreciated that the HELP menu could be omitted from the menubar 260 and the help files for the painting/drawing component could be accessed from the main HELP menu on the menubar 200. It will also be noted that when the painting/drawing component is invoked, the button 236 is grayed.

After a user creates a graphic image with the painting/drawing component, the addressing and mailing procedure is the same as sending an ordinary email. There is no need to save a file, encode it, or attach it to an email message. The main email component of the invention seamlessly performs all of the saving, encoding, and attaching without any of this being exposed to the user. See Appendices A, E and F. When the message is received by a person using a copy of the email program of the invention, the receiver's main email component seamlessly concatenates MIME parts, decodes the MIME file, determines that it is a message created with the painting/drawing component, invokes the painting/drawing component, and opens the message with the painting/drawing component. The receiver of the message does not have to download any file, find any attachment, execute any decoders, or launch any painting/drawing program to view/edit the graphic image created by the sender. See Appendices A, E and F.

Turning now to Figure 13, when the user clicks on the button 238, the spreadsheet component is invoked and it causes a new menubar 270, a grid 272, and a tool palette 274 to appear inside the message field 210 of the window 208. The spreadsheet component allows the author to create a spreadsheet and send it to another user for viewing/editing. Those skilled in the art will appreciate that the menubar 270 and palette 274 shown in Figure 13 contain the menus and tool icons typically found in a full featured spreadsheet program. Those skilled in the art will appreciate that the interface of the spreadsheet component shown in Figure 13 is similar to the interface of Microsoft® Excel® 98. It will be noted that the menubar 270 provides a separate HELP menu in addition to the HELP menu provided on the menubar 200. It will be appreciated that the HELP menu could be omitted from the menubar 270 and the help files for the spreadsheet component could be accessed from the main HELP menu on the menubar 200. It will also be noted that when the spreadsheet component is invoked, the button 238 is grayed.

After a user creates a spreadsheet with the spreadsheet component, the addressing and mailing procedure is the same as sending an ordinary email. There is no need to save a file, encode it, or attach it to an email message. The main email component of the invention seamlessly performs all of the saving, encoding, and attaching without any of this being exposed to the user. See Appendices A, E and F. When the message is received by a person using a copy of the email program of the invention, the receiver's main email component seamlessly concatenates MIME parts, decodes the MIME file, determines that it is a message created with the spreadsheet component, invokes the spreadsheet component, and opens the message with the spreadsheet component. The receiver of the message does not have to download any file, find any attachment, execute any decoders, or launch any spreadsheet program to view/edit the spreadsheet created by the sender. See Appendices A, E and F.

Turning now to Figure 14, when the user clicks on the button 240, the database component is invoked and it causes a new menubar 280, a record selection tool 282, and a free form space 284 to appear inside the message field 210 of the window 208. The database component allows the author to create a database and one or more reports and forms associated with the database and send it to another user for viewing/editing. Those skilled in the art will appreciate that the button bar 286 and the data fields 288 are defined by the author of the database using authoring tools found in the menus of the menubar 280. In fact, those skilled in the art will appreciate that the database interface shown in Figure 14 is similar to the interface of Filemaker®Pro 3.0. It will be noted that the menubar 280 provides a separate HELP menu in addition to the HELP menu provided on the menubar 200. It will be appreciated that the HELP menu could be omitted from the menubar 280 and the help files for the database component could be accessed from the main HELP menu on the menubar 200. It will also be noted that when the database component is invoked, the button 240 is grayed.

After a user creates a database with the database component, the addressing and mailing procedure is the same as sending an ordinary email. There is no need to save a file, encode it, or attach it to an email message. The main email component of the invention seamlessly performs all of the saving, encoding, and attaching without any of this being exposed to the user. See Appendices A, E, and F. When the message is received by a person using a copy of the email program of the invention, the receiver's main email component seamlessly concatenates MIME parts, decodes the MIME file, determines that it is a message created with the database component, invokes the database component, and opens the message with the database component. The receiver of the message does not have to download any file, find any attachment, execute any decoders, or launch any database program to view/edit the database created by the sender. See Appendices A, E, and F.

Turning now to Figure 15, when the user clicks on the button 242, the image editing component is invoked and it causes a new menubar 290 and a floating tool palette 292 to appear inside the message field 210 of the window 208. The image editing component allows the author to edit an image and send it to another user for viewing and/or further editing. Those skilled in the art will appreciate that the menubar 290 and palette 292 shown in Figure 15 contain the menus and tool icons typically found in a full featured image editing program. Those skilled in the art will appreciate that the interface of the image editing component shown in Figure 15 is similar to the interface of Adobe® Photoshop® 3.5. It will be noted that the menubar 290 provides a separate HELP menu in addition to the HELP menu provided on the menubar 200. It will be appreciated that the HELP menu could be omitted from the menubar 290 and the help files for the database component could be accessed from the main HELP menu on the menubar 200. It will also be noted that when the database component is invoked, the button 242 is grayed. Those skilled in the art will appreciate that image editing software is typically not used to create an image but to edit an image created by some other hardware/software such as a digital camera or a scanner. As such, there is typically a menu item for opening or capturing an image. As shown in Figure 15, open/capture commands may be found under the FILE menu in the menubar 200. Alternatively, image acquisition commands may be found under a menu item in the menubar 290.

After a user edits an image with the image editor component, the addressing and mailing procedure is the same as sending an ordinary email. There is no need to save a file, encode it, or attach it to an email message. The main email component of the invention seamlessly performs all of the saving, encoding, and attaching without any of this being exposed to the user. See Appendices A, E, and F. When the message is received by a person using a copy of the email program of the invention, the receiver's main email component seamlessly concatenates MIME parts, decodes the MIME file, determines that it is a message created with the image editor component, invokes the image editor component, and opens the message with the image editor component. The receiver of the message does not have to download any file, find any attachment, execute any decoders, or launch any image editor program to view/edit the image edited by the sender. See Appendices A, E, and F.

Turning now to Figure 16, when the user clicks on the button 244, the presentation (slide show) component is invoked and it causes a new menubar 300, a floating wizard palette 302, and a blank template 304 to appear inside the message field 210 of the window 208. The presentation component allows the author to create a slide show presentation and send it to another user for viewing and/or editing. Those skilled in the art will appreciate that the menubar 300, palette 302, and template 304 shown in Figure 16 are typical of those found in a full featured presentation program. In fact, those skilled in the art will appreciate that the interface of the presentation component shown in Figure 16 is similar to the interface of Microsoft® PowerPoint® 98. It will be noted that the menubar 300 provides a separate HELP menu in addition to the HELP menu provided on the menubar 200. It will be appreciated that the HELP menu could be omitted from the menubar 290 and the help files for the database component could be accessed from the main HELP menu on the menubar 200. It will also be noted that when the database component is invoked, the button 244 is grayed.

After a user creates a presentation with the presentation component, the addressing and mailing procedure is the same as sending an ordinary email. There is no need to save a file, encode it, or attach it to an email message. The main email component of the invention seamlessly performs all of the saving, encoding, and attaching without any of this being exposed to the user. See Appendices A, E, and F. When the message is received by a person using a copy of the email program of the invention, the receiver's main email component seamlessly concatenates MIME parts, decodes the MIME file, determines that it is a message created with the presentation component, invokes the presentation component, and opens the message with the presentation component. The receiver of the message does not have to download any file, find any attachment, execute any decoders, or launch any presentation program to view/edit the presentation created by the sender. See Appendices A, E, and F.

As described above, messages received by the email software according to the invention are seamlessly decoded and displayed. Figure 17 illustrates an incoming message window 408 which displays a message containing a combination of text and graphics in the message field 410. The incoming message window 408 also includes a subject field 412 and a "from:" address field 414 which includes information about the time the message was sent and received. Arrow buttons 416 allow the user to scroll through messages in the "in box". Button 418 drops a menu list of messages in the in box from which a message may be selected. The "+" button 420 adds the sender's address to the recipient's address book. The rotating arrow 427 hides the address field 414 and expands the message field 410. Buttons 428 and 430 are not implemented, but may be used for public key decryption, etc.

As mentioned above, the modular components of the invention may be authoring/reading components or read only components. Figure 17 illustrates an incoming message window 408 which displays a message containing a combination of text and graphics in message field 410 without any editing/authoring tools. The message may have been created with the word processing component or the painting and drawing component. The component used to create the message need not be known by the recipient of the message when it is opened with a read only component as shown. It will be appreciated that the message could also be automatically opened with an authoring/reading component, in which case, the message field 410 in Figure 17 would also include a menubar, and perhaps a tool palette. According to the invention, the email client software may be provided with a full complement of read only components and the authoring components may be installed according to the user's choices. Additionally, the email client software may be programmed to automatically download a reading component from an ftp site when it encounters a message which requires a component which is not yet installed.

There have been described and illustrated herein several embodiments of electronic mail software with modular integrated authoring/reading software components. While particular embodiments of the invention have been described, it is not intended that the invention be limited thereto, as it is intended that the invention be as broad in scope as the art will allow and that the specification be read likewise. Thus, while particular graphical interfaces have been disclosed, it will be appreciated that other interfaces could be utilized. Also, while particular authoring/reading components have been shown, it will be recognized that other types of authoring/reading components could be provided in the spirit of the invention. Moreover, while particular configurations have been disclosed in reference to the code in the appendices, it will be appreciated that other configurations could be used as well. Further, while particular software code and pseudocode have been disclosed to perform various functions, it will be appreciated that other code and/or hardware could be utilized to accomplish those functions and should be considered the equivalents thereof. It will therefore be appreciated by those skilled in the art that yet other modifications could be made to the provided invention without deviating from its spirit and scope as so claimed.

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
on startMovie

 global emG_passwordList, emG_userGroupList, emG_userGroup, emG_userName,
emG_registeredUsers, emG_msgNumber, emG_maildata, emG_mode, emG_noSimulate,
emG_mailFileList, emG_boxName

 --- Register the "YAK YAK" text to speech xtra
 -- register xtra "Yak", "XXXXXXXXXXXXX"

 -- VARIABLE LIST
 --- emG_userName: Tracks current user by name
 --- emG_msgNumber: Tracks if a message is new (empty) or old (number)
 --- emG_registeredUsers: Tracks users for to boxes in movies
 --- emG_passwordList: List of passwords for user logon: [password:name]
 --- emG_maildata: Message data list:
 ---        #to, #from, #re, #date, #mimetype, #mbxName, #msgbody
 -- NOT IMPLEMENTED -> #mbxName: now takes the place of #status - eliminate
case statement...
 --- emG_mode: flag for message movies; #author, #display
 --- emG_noSimulate: disable simulate Mode for message handler movies
 --- emG_userGroupList: for testing rebus game
 --- emG_userGroup: for testing rebus
 --- emG_mailFileList: List of locations of mailfiles for each user:
 ---              [uname:filename]
 --- emG_boxName: a mailbox datastructure; used to pass mailboxes to the mailbox
movie

 -- Install the menu
 installMenu "main menu"

 -- Clear all global variables

 set emG_noSimulate = TRUE

 --- Make sure the AddUsers button is not visible
 set the visible of sprite 20 = FALSE

 initSystemUsersData()
 initializeUser()
 initializeFields()
 fillStudentName()
 clearPassword()

end


on stopMovie

 global instanceOfXtra, emG_passwordList, emG_userGroupList, emG_userGroup,
emG_userName, emG_msgNumber, emG_maildata, emG_mode

 -- Clear all fields and global variables

 put "" into field "addPass"
 put "" into field "addUserGroup"
 put "" into field "addName"
 put "" into field "userList"
 put "" into field "studentName"
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
 put "" into field "studentUpName"
 put "" into field "studentPass"

 put "" into emG_userName
 set emG_msgNumber = 0
 set emG_registeredUsers = []
 set emG_passwordList = [:]
 set emG_userGroupList = [:]
 set emG_maildata = [:]

 set emG_userGroup = 0
 set emG_mode = #empty

 clearPassword()

 -- empty the script used to read in mailboxes
 set the scriptText of member 65 = ""

 --- Make sure the AddUsers button is not visible
 set the visible of sprite 20 = FALSE

end


-- score script 3 ss_goTheFrame

on exitFrame

 go the frame

end


--- Modified 8-9-98. To include a mailfile location for each
--- user. Added global variable emG_mailFileList. Also changed
--- format of the users file to be comma delimited items. This
--- will avoid problem with spaces in full pathnames for
--- user mailbox files.

on initSystemUsersData
 global emG_registeredUsers
 global emG_passwordList, emG_userGroupList, emG_mailFileList

 set emG_registeredUsers = []
 set emG_passwordList = [:]
 set emG_userGroupList = [:]
 set emG_mailFileList = [:]

 set usersData = readUsersFile()

 put the number of lines of usersData into totalLines
 repeat with i = 1 to totalLines

   if line i of usersData = EMPTY then
     nothing
   else
     set uname = item 1 of line i of usersData
     set pw = item 2 of line i of usersData
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
      set ugroup = value(item 3 of line i of usersData)
      set mfile = item 4 of line i of usersData

      add emG_registeredUsers, uname
      addProp emG_passwordList, uname, pw
      addProp emG_userGroupList, uname, ugroup
      addProp emG_mailFileList, uname, mfile
    end if

  end repeat

  sortRegisteredUsers()

end initSystemUsersData

-----------------------------------------------
on initializeUser

  global emG_userGroup, emG_userName
  global emG_msgNumber, emG_maildata, emG_mode

  put "" into emG_userName
  set emG_msgNumber = 0
  set emG_maildata = [:]
  set emG_userGroup = 0
  set emG_mode = #empty

end initializeMyGlobals

-----------------------------------------------
-- Initialize formatting of all visible text fields
-- Should be called when movie starts

on initializeFields

  -- SetTextInfo "StudentName", " ", "left", "arial", 14, "bold"
  SetTextInfo "StudentUpName", "your username here ", "left", "arial", 14, "bold"
  SetTextInfo "StudentPass", "", "left", "arial", 14, "bold"

  put "" into field "addPass"
  put "" into field "addUserGroup"
  put "" into field "addName"
  put "" into field "userList"

  -- set the lineHeight of field "To" = 18
  -- set the border of member "To" = 1
  -- set the border of member "ToDown" = 1
  -- set the margin of member "ToDown" to 8

end initializeFields

-----------------------------------------------
-- THIS HANDLER FILLS THE STUDENT LOGON NAME FIELD
-- WITH THE CURRENT LIST OF STUDENT NAMES

on fillStudentName
  global emG_registeredUsers

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
  -- Clear the student name field for the kids' logon
  put "" & RETURN into field "studentName"

  repeat with uname in emG_registeredUsers

    put uname & RETURN after field "studentName"

  end repeat

  -- Bring the field back to the top line
  set the scrollTop of member "studentName" = 0

end

-----------------------------------------------------------
-- For convenience of all the message handleing movies
-- keep emG_registeredUsers in a special sorted order:
-- alphabetic with "administrator" at the end.

on sortRegisteredUsers
  global emG_registeredUsers

  -- fix up emG_registeredUsers in sorted order but
  -- with "administrator" at the end

  deleteOne(emG_registeredUsers, "administrator")
  sort(emG_registeredUsers)
  append(emG_registeredUsers, "administrator")

end sortRegisteredUsers
--mailbox handlers

-------------------------------------------------------
--- openMailbox starts the mailbox movie
--- because the call must be continued in emh_continue
--- it is necessary to use a global variable for the
--- mailbox name.

on openMbx boxName
  global emG_boxName

  set emG_BoxName = boxName

  go to frame "movie"

  -- since all sprites are automatically puppets in Dir 6.0
  -- next should not be necessary
  -- Take control of the sidebar buttons

  puppetSprite 6, TRUE
  puppetSprite 7, TRUE
  puppetSprite 8, TRUE
  puppetSprite 9, TRUE

  set mbxMovie = window "mailbox.dir"
  set the titleVisible of mbxMovie to FALSE
  set the rect of mbxMovie = getMovieRect("mailbox")

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
 open mbxMovie
 set the name of mbxMovie to "childWindow"

 tell window "childWindow"
  -- next is a hack to get around Macromedia MIAW bug
  -- see emh_continue for calls to real handlers
  emc_startMeUp()

 end tell

 -- CONTINUES in emh_continue
end

------------------------------------------------
-- Read mailbox accepts a string that is the mailbox name
-- and returns a mailbox datastructure that is the
-- mailbox name and a list of the messages in that box

on readMailbox boxName
 global emG_userName, emG_mailFileList
--
-- "inbox" : set bxstring = "#mbxName: #received"
-- "outbox" : set bxstring = "#mbxName: #sent"
-- "savebox" : set bxstring = "#mbxName: #saved"
-- "trashbox" : set bxstring = "#mbxName: #trashed"

 set msgList = []


 set mbxStruc = list(boxName, msgList)
 set mailFileName = getProp(emG_mailFileList, emG_userName)

 -- Start up Fileio Xtra
 set instanceOfXtra = new(xtra "fileio")

 -- Set up Fileio to read from users file
 openFile(instanceOfXtra, mailFileName, 1)


 -- If file users doesn't exist, create it and set it up for read
 if status(instanceOfXtra) <> 0 then
  createFile(instanceOfXtra, mailFileName)
  openFile(instanceOfXtra, mailFileName, 1)
 end if

 -- Read what's currently in the file
 set whatText = readFile(instanceOfXtra)

 -- put msgs from appropriate box into the message list
 -- this needs to be fixed after the mail file datastructure
 -- is changed...

-- if value(#mbxname) <> 0 then
--  alert "Invalid mailbox name."
--  return(0)
-- end if

 --OLD case statement
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
    case boxname of
      "inbox" : set bxstring = "#status: #received"
      "outbox" : set bxstring = "#status: #sent"
      "savebox" : set bxstring = "#status: #saved"
      "trashbox" : set bxstring = "#status: #trashed"

      otherwise:
        alert "Invalid mailbox name."
        return(0)
    end case

  -- inefficient to have to look for the "#status...string"
  -- now is changed to value(#string) turning the string into a value, as
  -- Director has difficuties with strings w/in property lists...

  repeat with i = 1 to the number of lines in whatText

    if line i of whatText contains bxstring then
      append(msgList, value(line i of whatText))
    end if

  end repeat


  -- Close Fileio Xtra

  closeFile(instanceOfXtra)

  set instanceOfXtra = 0

  return(mbxStruc)

end

on messageHandler msgStatus

  global emG_userName, emG_maildata, emG_msgNumber, emG_mode,
emG_mailFileList

  put "" into sendData

  setProp emG_maildata, #status, msgStatus


  -- Set up where to find the users mailfile
  set whatFile = getProp(emG_mailFileList, emG_userName)


  -- Start up Fileio Xtra
  set instanceOfXtra = new(xtra "fileio")


  -- Set up Fileio to read and write from/to users file
  openFile(instanceOfXtra, whatFile, 0)


  -- If file users doesn't exist, create it and set it up for read/write
  if status(instanceOfXtra) <> 0 then
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
  createFile(instanceOfXtra, whatFile)
  openFile(instanceOfXtra, whatFile, 0)
end if


-- Read what's currently in the file
set whatText = readFile(instanceOfXtra)


-- Add message to current user's mailbox
--- if it previously existed, then write over the old message
--- if not, add it to the bottom
--- Only messages with a status = #saved can be changed.

if emG_msgNumber <> 0 then
  repeat with i = 1 to the number of lines in whatText
    if i = emG_msgNumber then
      put emG_maildata & RETURN after sendData
    else if line i of whatText <> "" then
      put line i of whatText & RETURN after sendData
    end if
  end repeat

else if emG_msgNumber = 0 then
  put whatText into sendData
  put emG_maildata & RETURN after sendData
end if


-- Put the cursor at the begining of the users file
setPosition(instanceOfXtra, 0)


-- Overwrite users file with updated list
writeString(instanceOfXtra, sendData)


-- Close Fileio Xtra

closeFile(instanceOfXtra)

set instanceOfXtra = 0


-- ON SEND, PUT IN OTHER CHILD'S MAILBOX, TOO

if msgStatus = #sent then

  setaProp emG_maildata,#status,#received

  put getaProp(emG_maildata,#to) into sendingTo

  put "" into sendData


  -- Set up where to find the users file.
  -- put the pathName & sendingTo into whatFile
  set whatFile = getProp(emG_mailFileList, sendingTo)
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```


  -- Start up Fileio Xtra
  set instanceOfXtra = new(xtra "fileio")


  -- Set up Fileio to read and write from/to users file
  openFile(instanceOfXtra, whatFile, 0)


  -- If file users doesn't exist, create it and set it up for read/write
  if status(instanceOfXtra) <> 0 then
   createFile(instanceOfXtra, whatFile)
   openFile(instanceOfXtra, whatFile, 0)
  end if


  -- Read what's currently in the file
  set whatText = readFile(instanceOfXtra)

  -- Add message to recipient's mailbox
  put emG_maildata & RETURN after whatText

  -- Put the cursor at the begining of the users file
  setPosition(instanceOfXtra, 0)

  -- Overwrite users file with updated list
  writeString(instanceOfXtra, whatText)


  -- Close Fileio Xtra

  closeFile(instanceOfXtra)
  set instanceOfXtra = 0

 end if

end
------------------------------------------------
on createMailData userName, type

 set newmsg = [:]
 addProp(newmsg, #to, "")
 addProp(newmsg, #from, userName)
 addProp(newmsg, #re, "")
 addProp(newmsg, #date, the abbreviated date)
 addProp(newmsg, #mimetype, type)
 addProp(newmsg, #status, #new)
 addProp(newmsg, #msgbody, [])
 return(newmsg)

end createMailData

----------------------------------------
--- Make sure there is something in each of the "to"
--- and "from" fields and that the messagebody has the
--- right format.

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
on isValidMessage maildata

  repeat with prop in [#to, #from]
    if getProp(maildata, prop) = "" then
      alert "But who do you wish to send this message to?"
      --return(0)
    end if
  end repeat

  if not listp(getProp(mailData, #msgBody)) then return(0)

  return(1)

end isValidMessage


---------------------------------------------

on setReply

  -- TAKES CARE OF SWITCHING THE SIDEBAR BUTTONS WHEN REPLY
  -- IS HIT FROM AN OPEN MESSAGE

  go to "Movie"  -- make sure the frame is correct

  -- Set the buttons with reply off and send on
  disableReply()
  enableSend()

end

---------------------------------------------

on disableSend
  go to "movie"
  puppetsprite 7, TRUE
  set the member of sprite 7 = member "SendNo"
end disableSend

on enableSend
  go to "movie"
  puppetsprite 7, TRUE
  set the member of sprite 7 = member "Send"
end enableSend

on disableReply
  go to "movie"
  puppetsprite 6, TRUE
  set the member of sprite 6 = member "ReplyNo"
end disableSend

on enableReply
  go to "movie"
  puppetsprite 6, TRUE
  set the member of sprite 6 = member "Reply"
end enableSend
```

```
on returnToMain

 global emG_msgNumber, emG_maildata, emG_mode

 -- Clear the variables

 set emG_msgNumber = 0
 set emG_maildata = [:]
 set emG_mode = #empty

 --- unpuppet the left panel buttons which reuse sprite
 --- channels 6-9
 -- MB 10-13-98 I don't like this method... it is safer
 -- to use new sprite channels.
 -- is there a good reason for reusing channels...does it
 -- affect performance?

 puppetsprite 6, FALSE
 puppetsprite 7, FALSE
 puppetsprite 8, FALSE
 puppetsprite 9, FALSE

 -- Go back to the main menu

 go to "open"

end

-- API handlers

--- emh_getUserMailbox returns the current user's mailbox specified
--- by the mailBoxName parameter.

on emh_getUserMailbox mailboxName

 return(readMailbox(mailBoxName))

end emh_getUserMailbox

---------------------------------------------------------

on emh_getUserName
 global emG_userName

 return(emG_userName)

end emh_getUserName

---------------------------------------------------------

on emh_getUserData userName
 global emG_userGroupList, emG_mailFileList

 return(list (username, ¬
 username, getProp(emG_userGroupList, username), getProp(emG_mailFileList,
userName), [], list ( "inbox", "outbox", "savebox") ))

end emh_getUserData
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```


-- more API handlers

---------------------------------------------
--- A curse on Macromedia.  This ugly hack is used to get
--- around a Macromedia bug which causes the startMovie
--- handler of a MIAW to run only after control has been
--- transferred back to the calling movie and the calling
--- movie advances a frame.

--- This handler is called by the startMovie handler of the
--- MIAW.  This way we ensure that these scripts only run
--- after the MIAW has been properly initialized.

on emh_continue componentType
  global emG_userName, emG_maildata, emG_mode, emG_boxName, emG_userGroup

  -- Since this function can only be called by a MIAW component
  -- we assume that the "childwindow" is running

  if componentType = #msgHandler then
    tell window "childwindow"
      emc_initWindow(emG_userName)
      msh_openMessage(emG_maildata, emG_mode)
    end tell

  else if componentType = #mailbox then
    tell window "childwindow" to emc_initWindow(emG_userName)
    set success = the result
    if not success then
      alert "Could not initialize mailbox movie"
      forget window "childwindow"
      return(0)
    end if

    set mbx = readMailbox(emG_boxName)
    tell window "childwindow" to mbx_openMailbox(mbx)
    set success = the result
    if not success then
      alert "Could not open mailbox."
      forget window "childwindow"
      return(0)
    end if

  else alert "ERROR invalid componentype."

end emh_continue


-- more API handlers
---------------------------------------------
-- The emh_passMessage handler is used to pass a message from
-- a mailbox to the appropriate message handler

on emh_passMessage maildata, messageNumber

  global emG_maildata, emG_msgNumber, emG_mode
```

```

-- should check for errors in the parameters

set emG_maildata = mailData
set emG_msgNumber = messageNumber

-- If a mailbox window is open we need to close that window.
-- The window will not actually close until this function completes
-- and returns control to the caller function in the mailbox movie.
-- Therefore, we need to move it to the back so it is no longer visible.

moveToback window "childwindow"
updatestage

tell window "childWindow" to emc_getComponentInfo()
set cInfo = the result
if getComponentProp(cInfo, #ComponentType) = #mailbox then
  tell window "childWindow" to emc_closeWindow()
  forget window "childWindow"
end if

go to frame "movie"
-- set up the button bar on the left

set msgStatus = getProp(emG_maildata, #status)
if msgStatus = #received then    -- from inbox
  set emG_mode = #display
  disableSend()
  enableReply()
else if msgStatus = #sent then   -- from outbox
  set emG_mode = #display
  disableSend()
  disableReply()
else if msgStatus = #saved then  -- from savebox
  set emG_mode = #author
  disableReply()
  enableSend()
else  -- error
  alert "passing message with invalid status"
  return(0)
end if

--- OPEN MESSAGE HANDLER MOVIE

openMsgHandler(getaProp(emG_maildata,#mimetype), emG_maildata)

end emh_passMessage

-- more API handlers
-----------------------------------------------------
-- THIS CODE IS BASED ON OLD STUFF WHICH USES MESSAGE NUMBER
-- TO IDENTIFY MESSAGES ACROSS MAILBOXES.  THIS SYSTEM
-- NEEDS TO BE CHANGED TO IDENTIFY MESSAGES BY A MAILBOXNAME
-- AND A MESSAGE NUMBER WITHIN THE BOX

on emh_getMessage messageNumber, typeorBoxName

 global emG_userName, emG_msgNumber, emG_mailData
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

set emG_msgNumber = messageNumber

if messageNumber = 0 then  -- return new message data
  --typeorBoxName should have mimetype
  set emG_maildata = createMailData(emG_userName, typeorBoxName)
  return(emG_maildata)
end if

-- otherwise find an existing message
-- typeorboxname should have boxName

set theBox = readMailbox(typeorBoxName)
set emG_mailData = getat(getAt(theBox, 2), messageNumber)
return(emG_maildata)

end emh_getMessage

-----------------------------------------------------------

on emh_getRegisteredUsers
  global emG_registeredUsers

  return(emG_registeredUsers)

end emh_getRegisteredUsers

-----------------------------------------------------------

on emh_killComponent

  tell window "childwindow" to emc_closeWindow()
  if the result = 0 then alert "TROUBLE CLOSING WINDOW!"
  else
    forget window "childwindow"
  end if
  returnToMain()

end emh_killComponent

-----------------------------------------------------------
--- Initialize formatting of text fields
--- Thanks to Frank Leahy, maricopa site for this one

on SetTextInfo fldName, fldValue, fldAlign, fldFont, fldSize, fldStyle

  put fldValue into field fldName
  set the textAlign of field fldName = fldAlign
  set the textFont of field fldName = "arial"   --fldFont
  set the textSize of field fldName = fldSize
  set the textStyle of field fldName = fldStyle

end

-----------------------------------------------------------

-- script of cast member studentName
-- emG_userName should not be set here
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
-- because it could be invalid

on mouseUp

 -- Put selected user name into up version of student field
 -- switch the field from down to up

 put word 1 of line(the mouseLine) of field "studentName" into field "studentUpName"

 set the member of sprite 14 to member "StudentUpName"

end


-- script of cast member studentUpName

on mouseUp

  -- Pull down student field: change field from
  -- up (sprite 17) to down (sprite 16)

  set the member of sprite 14 to member "StudentName"

  -- clear password field
  clearPassword()

end

----------------------------------------------------------------------

-- scripts of cast member studentPassword


on keyUp
  global gpw, gpwlen
  --gpw is global password and
  --gpwlen is global password length

  hideAlert() -- user maybe trying again...hide badPwMsg

  if the key = RETURN then
    if checkPassword(field "studentUpName", gpw) then
      enterMainEmail(field "studentUpName")
    else --- invalid password
      alertBadPassword()
    end if
    set gpw = ""
    set gpwlen = 0
    put "" into field "studentPass" -- reset the password field
  end if

end keyUp

-------------------------------------

on keyDown
  global gpwlen, gpw

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
--eats the key, otherwise it will appear until keyup

if the key = BACKSPACE then
  put "" into char gpwlen of field "studentPass"
  put "" into char gpwlen of gpw
  if gpwlen > 0 then
    set gpwlen = gpwlen - 1
  end if
else if the key = RETURN then
  nothing
else if the keycode >= 117 and the keycode <= 126 then
  nothing
else
  put "*" after field "studentPass"
  put the key after gpw
  set gpwlen = gpwlen + 1

end if

set the selstart =  gpwlen
set the selend = the selstart

end keyDown


-- script of cast member goStudentLog

on mouseUp

 go to frame "pass"

end


-- script of cast member editUsers

on mouseUp

  -- set the default pathname for the mail file location
  put the pathname into field "addMailFileLoc"

  go to frame "edit"

end


-- script of cast member okUser


on mouseDown
  set the member of sprite 7 = "okay down"
end



-- script of cast member okDown


```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
on mouseUp
  global gpw, gpwlen  --- see script of field studentPass

  set the member of sprite 7 = "okayUser"

  if checkPassword(field "studentUpName", gpw) then
    -- valid user & pw
   enterMainEmail(field "studentUpName")

  else  -- password invalid

   alertBadPassword()

  end if

  clearPassword()

end
-- script of cast member addUser

on mouseUp
  global emG_registeredUsers
  global emG_passwordList, emG_userGroupList, emG_mailFileList

  --check that username is filled and is unique

  if field "addName" = EMPTY then
    alert "No username"
    return(0)
  else if getOne(emG_registeredUsers, field "addName") then
    alert "Username already in system.  Choose a different name"
    return(0)

  else set uname = field "addName"

  --NEED TO TAKE CARE OF THIS!!!!
  -- check that the mailfile location is a valid directory
  -- there are serious problems with this at present
  -- for now assume pathnames are valid


  --  add new User data to system global variables
  add(emG_registeredUsers, uname)
  addProp(emG_passwordList, uname, field "addPass")
  addProp(emG_userGroupList, uname, field "addUserGroup")
  -- append username to the mailfile location directory
  addProp(emG_mailFileList, uname, field "addMailFileLoc" & uname)

  sortRegisteredUsers()

  -- write the users file with system users data
  writeUsersFile()

  --  Put the updated user list into the userList field
  put "" into field "userList"
  repeat with uname in emG_registeredUsers
    put uname after field "userList"
    put " " & getProp(emG_passwordList, uname) after field "userList"
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
    put " " & getProp(emG_userGroupList, uname) after field "userList"
    put " " & getProp(emG_mailFileList, uname) after field "userList"
    put RETURN after field "userList"
  end repeat


  -- reset the User data fields

  put "" into field "addUserGroup"
  put "" into field "addPass"
  put "" into field "addName"
  put the pathname into field "addMailFileLoc"

  -- Refill the kids' logon name field
  fillStudentName()

end

-- script of cast member seeUserList

on mouseUp

  global instanceOfXtra


  put "" into field "userList"


  -- Set up where to find the users file
  put the pathName & "users" into whatFile


  -- Start up Fileio Xtra
  set instanceOfXtra = new(xtra "fileio")


  -- Set up Fileio to read from users file
  openFile(instanceOfXtra, whatFile, 1)


  -- If file users doesn't exist, create it and set it up for read to avoid error

  if status(instanceOfXtra) <> 1 then
    createFile(instanceOfXtra, whatFile)
    openFile(instanceOfXtra, whatFile, 1)
  end if


  -- Read what's currently in the file
  set whatText = readFile(instanceOfXtra)


  -- Put the updated user list into the userList field
  put whatText into field "userList"

  -- Close Fileio Xtra
  closeFile(instanceOfXtra)

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
  set instanceOfXtra = 0

end



-- script of cast member DoneAdmin

on mouseUp

  go to frame "open"

  put "" into field "addName"
  put "" into field "addUserGroup"
  put "" into field "addPass"
  put "" into field "addMailFileLoc"

end
-- msgHandlers scripts
-------------------------------------------------
--- openMsgHandler starts the appropriate Message Handling movie.
--- The call must be continued in emh_continue.
--- It is necessary that the global variable emG_mailData is
--- set up. Therefore, we pass it as a parameter to make it
--- clear that the variable is necessary.

on openMsgHandler mimetype, mailData

  set movieName = getMessageHandler(mimetype)
  go to frame "movie"

  -- since all sprites are automatically puppets in Dir 6.0
  -- next should not be necessary
  -- Take control of the sidebar buttons

  puppetSprite 6, TRUE
  puppetSprite 7, TRUE
  puppetSprite 8, TRUE
  puppetSprite 9, TRUE

  set mshMovie = window movieName
  set the titleVisible of mshMovie to FALSE
  set the rect of mshMovie = getMovieRect(mimetype)

  open mshMovie
  set the name of mshMovie to "childWindow"

  tell window "childWindow"
    -- next is a hack to get around Macromedia MIAW bug
    -- see emh_continue for calls to real handlers
    emc_startMeUp()

  end tell

  -- CONTINUES in emh_continue
end openMsgHandler


```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
-----------------------------------------------
-- getMessageHandler returns filename of movie to handle mimetype.
-- This code makes it easy to make changes in movie filenames
-- and to add new message handling movies.

on getMessageHandler mimetype

 case mimetype of
   "text":  return("text.dir")
   "rebus": return("rebus.dir")
   "grid": return("grid.dir")
   "connect": return("connect.dir")
   "puzzle" : return("puzzle.dir")

   otherwise:
     alert "Invalid mimetype of message."
     return("")
 end case

end getMessageHandler
-----------------------------------------------

on getMovieRect whichMovie

 --- the top of green panel
 set movieTop = the top of sprite 3
 --- the left of green panel
 set movieLeft = the left of sprite 3

 case whichMovie of
   "rebus", "rebus.dir":
     set theRect= rect(movieLeft, movieTop, ¬
                      movieLeft + 640, movieTop +480)
   "text", "text.dir":
     set theRect= rect(movieLeft, movieTop, ¬
                      the stageRight - 5, the stageBottom -5)
   "puzzle", "puzzle.dir":
     set theRect= rect(movieLeft, movieTop, ¬
                      the stageRight - 5, the stageBottom -5)
   "grid", "grid.dir", "connect", "connect.dir":
     set theRect= rect(movieLeft, movieTop, ¬
                      the stageRight - 5, the stageBottom -5)
   "mailbox", "mailbox.dir":
     set theRect= rect(movieLeft, movieTop, ¬
                      the stageRight - 5, the stageBottom -5)
   otherwise:
     alert "ERROR: invalid movieName: " & whichMovie
     set theRect = rect(0,0,0,0)

 end case

 return(theRect)

end getMovieRect



-- score script  fr_installMenu
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

on prepareFrame
  --first clear away any old menus
  installMenu 0
  installMenu "main menu"
end

------------------------------------------------------------------------
-- password verification and user init



on enterMainEmail username
  global emG_userName, emG_userGroup, emG_userGroupList

  set emG_userName = username
  set emG_userGroup = getProp(emG_userGroupList, emG_userName)

  -- ADMINISTRATOR has access to the "Edit Users" button
  if emG_userName = "administrator" then
    set the visible of sprite 20 = TRUE
  end if

  go to frame "open"

end enterMainEmail

-----------------------------------------------------


on checkUserName userName
  global emG_registeredUsers

  if getone(emG_registeredUsers, userName) then
    return(1)   -- username is in system

  else
    alert "User " & userName & "not a KidCode authorized user." & RETURN & "You
cannot login without a valid user name."

  end if

end checkUsername

-- more password handling scripts

on checkPassword userName, password
  global emG_passwordList

  -- if the username is not valid quit this...
  if not checkUserName(userName) then return(0)

  -- username is valid

  --  First part of loop changes capital letters to lower case
  --  Second part puts lower case letters into password check
  --  This eliminates all spaces and/or unacceptable characters

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
set checkPassword = ""
repeat with i = 1 to the number of chars in password

  put char(i) of password into capital
  put charToNum(capital) into capital

  if capital <= 90 and capital >= 65 then
    put numToChar(capital + 32) after checkPassword
  else if capital >= 97 and capital <= 122 then
    put numToChar(capital) after checkPassword
  end if

end repeat

-- CHECK PASSWORD

set realPassword = getProp(emG_passwordList, username)

if realpassword = checkPassword then
  return(1)  --TRUE
else
  return(0)
end if


end checkPassword

--------------------------------------------

on clearPassword
  global gpw, gpwlen

  set gpw = ""
  set gpwlen = 0
  put "" into field "StudentPass"

end clearPassword

--------------------------------------------

on alertBadPassword

  set the loc of sprite 17 to point(231, 350)
  beep()

end alertBadPassword
on hideAlert

  set the loc of sprite 17 to point(-188, -31)

end hideAlert


-- script of cast member reply

on mouseUp
  global emG_userName, emG_msgNumber
  global emG_maildata, emG_mode, emG_userGroup
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

-- abandon current MailData which should be in the inbox.
-- Later, the user may choose to either abandon or send
-- the new replyTo message. That is not a concern.

-- If a mailbox window is open need to get the message
-- and close that window.

tell window "childWindow" to emc_getComponentInfo()
set cInfo = the result
if getComponentProp(cInfo, #ComponentType) = #mailbox then
  tell window "childwindow" to mbx_GetMessageNumber()
  set emG_msgNumber = the result
  if emG_msgNumber <= 0 then
    alert "You must select a message."
    return()  -- abandon the request to reply
  end if

  tell window "childwindow" to mbx_GetMessage(emG_msgNumber)
  set emG_maildata = the result

  --- forget window "childwindow" -- done in passMessage

  --- Now open the appropriate Message Handler
  --- to display the message

  emh_PassMessage(emG_maildata, emG_msgNumber)

end if

-- If we got to this point message handler is open.
-- Presumably it has a message displayed. If the message
-- is empty only the message handler knows that and it
-- will need to catch the error and return an error code
-- to msh_replyMessage.

-- The message handling movie's replyMessage handler
-- should swap "to" and "from"
-- fields and make the message editable

-- set mode to author to keep it consistent with msg handler
set emG_mode = #author

set emG_msgNumber = 0  -- this is now a new message

tell window "childWindow"
  global emG_userGroup
  -- msg handler will swap "to" with "from" and change
  -- mode to author
  moveToFront window "childWindow"
  msh_replyMessage()
end tell

set emG_maildata = the result

-- Toggle the send and reply buttons
setreply  -- disable reply and enable send buttons

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
end


-- script of cast member send

on mouseUp
  global emG_maildata, emG_userGroup

  -- Could check that the childwindow is a messagehandler
  -- but this may not be necessary.

  tell window "childWindow"
    global emG_userGroup
    msh_sendMessage()
    set emG_maildata = the result
  end tell

  if not isValidMessage(emG_maildata) then
    alert "ERROR not a valid message."
    return(0)         -- abandon attempt to send
  end if

  --- otherwise continue to send message

  -- NEED TO FIX THIS SO THAT MESSAGE STATUS DOES NOT
  -- BECOME "#sent" if it fails to be saved to both
  -- mail files

  messageHandler(#sent)  -- for now this uses global emG_maildata

  -- tell window "childWindow" to msh_clearMessage()

end

-- script of cast member print

on mouseUp

  tell window "childwindow" to emc_getComponentInfo()
  set cInfo = the result
  set cType = getComponentProp(cInfo, #ComponentType)

  if cType = #mailbox then
    -- need to pass the message to its message handling
    -- component for printing.  Ideally this can be done
    -- without opening a window and laying out the message.

    alert "I can't do that right now. Open the message and then print."

  else if cType = #msgHandler then

    tell window "childwindow"
      msh_PrintMessage()
    end tell

  else alert "ERROR invalid componentype."

```

```
end

---------------------------------------------------------------
-- script of cast member Quit

on mouseUp

 handleQuit()

end

---------------------------------------------------------------

on handleQuit

 initializeUser()
 clearPassword()
 go to frame 2

 -- make sure the editUsers button is invisible
 set the visible of sprite 20 = FALSE

end handleQuit

-- script of cast member trash

--- Email Main now handles all aspects of trashing a
--- message by writing the mail files.  The components
--- are instructed to update their state by clearing the
--- message (if the component is a message handler) or
--- redrawing the message list (if the component is a
--- mailbox.)

--- Should add a confirmation dialog with the user

on mouseUp
 global emG_msgNumber -- number of the current message

 tell window "childwindow" to emc_getComponentInfo()
 set cInfo = the result
 set cType = getComponentProp(cInfo, #ComponentType)


 if cType = #mailbox then
   -- need to determine which message(s) are currently
   -- selected and instruct the mailbox to update its
   -- display

   -- temporary implementation of mbx_trashMessages does
   -- not handle multiple messages as a result the
   -- arguments are ignore...

   tell window "childwindow" to mbx_trashMessages([])

   -- the following lines will be neceessary when
   -- mbx_trashMessages is properly implemented.  For
   -- now, the temporary implementation trashes the
   -- message itself.
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
   -- set messageNumbers = the result
   -- delete each message in the list of messageNumbers


  else if cType = #msgHandler then

   -- rewrite the message into the mailfile
   messageHandler(#trash)

   tell window "childwindow" to msh_clearMessage()

  else alert "ERROR invalid componentype."


end


-- script of cast member text

on mouseUp
  global emG_msgNumber
  global emG_maildata, emG_mode

  -- START A NEW MESSAGE

  set emG_msgNumber = 0
  set emG_mode = #author
  set emG_maildata = createMailData(emG_userName, "text")

  openMsgHandler("text", emG_mailData)

  disableReply()

end

-----------------------------------------------------------------
-- script of cast member Rebus

on mouseUp
  global emG_msgNumber
  global emG_maildata, emG_mode

  -- START A NEW MESSAGE

  set emG_msgNumber = 0
  set emG_mode = #author
  set emG_maildata = createMailData(emG_userName, "rebus")

  openMsgHandler("rebus", emG_mailData)

  disableReply()

end


--- script of cast member grid

on mouseUp
```

```
  global emG_msgNumber
  global emG_maildata, emG_mode

  -- START A NEW MESSAGE

  set emG_msgNumber = 0
  set emG_mode = #author
  set emG_maildata = createMailData(emG_userName, "grid")

  openMsgHandler("grid", emG_mailData)

  disableReply()

end


--- script of cast member puzzle

on mouseUp
  global emG_msgNumber
  global emG_maildata, emG_mode

  -- START A NEW MESSAGE

  set emG_msgNumber = 0
  set emG_mode = #author
  set emG_maildata = createMailData(emG_userName, "puzzle")

  openMsgHandler("puzzle", emG_mailData)

  disableReply()

end


--- script of cast member connect

on mouseUp
  global emG_msgNumber
  global emG_maildata, emG_mode

  -- START A NEW MESSAGE

  set emG_msgNumber = 0
  set emG_mode = #author
  set emG_maildata = createMailData(emG_userName, "connect")

  openMsgHandler("connect", emG_mailData)

  disableReply()

end

------------------------------------------------------------
on getComponentProp infoList, prop

  --- need to add error checking code

```

```
 case prop of
  #componentName: return(getAt(infolist, 1))
  #componentID: return(getAt(infolist, 2))
  #componentType: return(getAt(infolist, 3))
  #componentMIMEtype: return(getAt(infolist, 4))

  otherwise: alert "ERROR no component property."
 end case

end getComponentProp

-- script of cast member savebox

on mouseUp

 openMbx("savebox")

end


-- script of cast member inbox

on mouseUp

 openMbx("inbox")

end



-- script of cast member outbox

on mouseUp

 openMbx("outbox")

end


--- Users File functions

-- returns a string of all users data from the users file.

-- THIS FUNCTION NEEDS TO CHECK THAT DATA IS VALID

on readUsersFile

  -- Set up where to find the users file
  put the pathName & "users" into whatFile

  -- Start up Fileio Xtra
  set instanceOfXtra = new(xtra "fileio")

  -- Set up Fileio to read from users file
  openFile(instanceOfXtra, whatFile, 1)


  -- If file users doesn't exist, create it

 if status(instanceOfXtra) <> 0 then
   createFile(instanceOfXtra, whatFile)
   openFile(instanceOfXtra, whatFile, 1)
 end if


 -- Read what's currently in the file
 set whatText = readFile(instanceOfXtra)


 -- if no users are defined, assume administrator as default user
 -- Administrator info is not written into the user's file until at
 -- least one user is defined. This occurs in AddUsers functions.

 if whatText = "" then
   -- for now, assume admin has mail file in each
   -- location where kidcode is installed
   put "administrator,kidcode,0," & the pathName & "administrator" & RETURN into
whatText
 end if

 -- Close Fileio Xtra
 closeFile(instanceOfXtra)
 set instanceOfXtra = 0

 return(whatText) -- string read from users file

end readUsersFile


-------------------------------------------------------
-- more users file scripts

on writeUsersFile
  global emG_registeredUsers, emG_passwordList, emG_userGroupList,
emG_mailFileList

  -- Set up where to find the users file
  put the pathName & "users" into whatFile

  -- Start up Fileio Xtra
  set instanceOfXtra = new(xtra "fileio")

  -- Set up Fileio to read and write from/to users file
  openFile(instanceOfXtra, whatFile, 0)

  -- If file users doesn't exist, create it and set it up for read/write

  if status(instanceOfXtra) <> 0 then
    createFile(instanceOfXtra, whatFile)
    openFile(instanceOfXtra, whatFile, 0)
  end if

  -- Put the cursor at the begining of the users file
  setPosition(instanceOfXtra, 0)

  --- put together string of usersData
 set whatText = ""
 repeat with uname in emG_registeredUsers

  set pw = getProp(emG_passwordList, uname)
  set ugroup = getProp(emG_userGroupList, uname)
  set mfile = getProp(emG_mailFileList, uname)
  set whatText = whatText & uname & "," & pw & "," & ugroup & "," & mfile &
RETURN

 end repeat

 -- Overwrite users file with updated list
 writeString(instanceOfXtra, whatText)

 -- Close Fileio Xtra

 closeFile(instanceOfXtra)

 set instanceOfXtra = 0
 return(1)

end writeUsersFile

--------------------------------------------------
--- these next functions are created to do file checking
--- however they appear to suffer from severe crash problems
--- these problems will also effect mail file creation if
--- path names are invalid...we need to fix this

on pathp pathname

 set instanceOfXtra = new(xtra "fileio")
 openFile(instanceOfXtra, pathname, 1)
 set theval = status(instanceofxtra)

 case theval of
   0 :
    closeFile(instanceOfXtra)
    set instanceOfXtra = 0
    return(1)

   -36: -- I/O Error...likely to cause system crash
    alert "System has become unstable. " & RETURN & "Please save your work."
    -- next call to fileio xtra may crash system
    set instanceOfXtra = 0
    return(0)

   otherwise :
    alert " " & error(instanceOfXtra, theval)
    closeFile(instanceOfXtra)
    set instanceOfXtra = 0
    return(0)

 end case

 end pathExists
```

```
on **foldertest**
  getNthFileNameInFolder("C:\windows", 1)
  end foldertest
```

App ndix B: KidCode® API

KidCode® Application Programming Interface (API)

This API defines the data and function calls that are used for communication between the KidCode Main Email program and installable components. Each installable component can be one of two types:

- mailbox browser/editor component
- message authoring/display component

KidCode Main Email application may communicate with another mail server such as an SMTP compliant server to retrieve and store email messages. Alternatively, the Email Main program may include code for many of the functions normally associated with a mail server program. Whether in conjunction with a mail server, or on its own, the Email Main program handles all functions associated with sending and receiving email messages. This includes reading and writing mailbox files to/from permanent storage or other mail servers on a network (e.g. using POP3), finding and verifying network addresses, and sending mail messages to other servers on a network.

The Main Email Program also provides a GUI that provides interaction with a user for those functions that are directly associated with storage and transfer of electronic mail messages and mailboxes. In particular, the Main Email program includes buttons and/or menu items that allow a user to:

- **Send** (a message),
- **Reply** (to a message),
- **Open** (a message or a mailbox),
- **Delete/Trash** (messages or mailboxes),
- **Save** (a message to an alternative mailbox)
- **Print** (a message)

The Main Email Program also handles all data bundling and unbundling that may be required to transform the message data used by a message authoring component into a fully MIME compliant message type. This way each message authoring component can handle data in a format most convenient to it and all MIME parsing and details associated with protocol compliance can be centralized in the Main Email application. The only requirement for the message data passed between a message authoring component and the Main Email Program is that the message body data be formatted either as an ASCII string or in a binhex format.

The KidCode Main Email program communicates with installable components in order to execute the commands defined above.

Mailbox browser/editor components
Mailbox components are used to display, edit, and browse mailboxes. Different kinds of users and different types of messaging applications (e.g. fax, traditional email, internet voice) may require very different displays and functionality from a mailbox viewer/editor. Installable mailbox components make it possible to upgrade, select from multiple viewing formats, utilize different mailbox viewer/editors for different users, and in general increase the range of functionality that can be achieved within one basic messaging application program.

Message authoring/display components
Message handler components make it possible to handle an unlimited number of message types. Each message handler component is designed to deal with a specific MIME type of message. The MIME data standard has been designed so that application developers can define new MIME types as needed by labeling these with the "/application-x" prefix. A message handler component can be any program that defines a message MIME type of

data that it handles and that implements the callback entry points described in this document. These functions allow the Main Email application to obtain information about the message handler and allows the message handler to respond to standard mail commands such as Send or Reply, that have been issued by a user through the Main Email interface. Example message handler components included in the KidCode application are an ordinary ascii text message handler, a game called Rebus that allows users to create and respond to graphical rebus messages, an a sample mathematics workbook that allows students and a teacher to send workbook problems to one another.

Global variable naming conventions:

Each movie should name its global variables with a prefix that identifies the movie and a capital "G" for "global". We will keep track of each movie's prefix. For now we have the following identifing prefixes:

component prefix	component	global variable prefix
em_	main movie	emG_
tm_	text movie	tmG_
rm_	rebus movie	rmG_
cm_	connect movie	cmG_
tgm_	text grid movie	tgmG_
pm_	puzzle movie	pmG_
mbx_	mailbox movie	mbxG_

Main Movie Public Data Types

```
em_ComponentType  symbol = #mailbox or #msgHandler

em_UserName  string

em_UserData struct (
                str        UserName
                str        FullName
                str               ReturnAddress
        em_AddressBook        AddressBook
        em_MailboxList        Mailboxes
                str               SMTPHost
                str               POP3Host
                str               Password
)

em_MailboxName  string

em_Mailbox  struct (
        em_mailboxName  boxName
        list of emMailData
)

em_RegisteredUsers  list of em_UserName
```

Appendix B: KidCode® API

```
em_MailData struct (
        em_Address          To
        em_Address          From
                str         Re
                str         Data
                str         MimeType
                list        MsgBody
)

em_MessageNumber int

em_Mode symbol = #author or #display

em_ComponentInfo struct (
                str         ComponentName
                int         ComponentID
em_ComponentType            ComponentType
                str         ComponentMIMEType  ; nil if mailbox
)
```

Email Main API Functions

These functions are called by the installable components to access services provided in the KidCode Main Email program.

```
/
*****************************************************************
****/
/* emh_getUserMailbox
Return a mailbox data structure for the current user and mailbox name. This function is
normally called by a mailbox handling component. Mailbox handling components may
use temporary files to hold mailbox contents but they should never access the users
mailbox files. All access to these files must be obtained through the Main Email
program.
*/

em_Mailbox  emh_getUserMailbox (
                em_MailBoxName
)


/
*****************************************************************
****/
/* emh_getUserData
Return a data structure with user information. The KidCode Main Email program
maintains all user information and handles user administration functions. The Main
program also communication with external Mail servers which may contain other user
information not part of the KidCode API.
*/

em_UserData emh_getUserData (
        em_UserName,
)


/
****************************************************************
****/
/* emh_continue
Used by installable components to explicitly pass control back to the Main Email
program. This function is necessary for the Director/Lingo implementation.
*/

void emh_continue (
        em_ComponentType
)


/
****************************************************************
****/
/* emh_killComponent
Used by an installable component to inform the Main Email program that it is preparing
to terminate. This allows the Main program to free any memory and/or data structures that
have been allocated to the component.
*/

void emh_killComponent (
)


/
****************************************************************
****/
/* emh_passMessage
Used primarily by mailbox components to pass a message to the Main program so that it
can be displayed by the appropriate message handling component. Email main takes the
message argument (em_MailData, looks up the Mimetype of the message, and invokes the
appropriate message handler to display the message.
*/

void emh_passMessage (
        em_MailData,
        em_MessageNumber
)


/
****************************************************************
****/
/* emh_getMessage
Returns the message (em_MailData) with Number MessageNumber from the
MailboxName of the current user. Can be used by installable components to retrieve
specific messages from the user's mailboxes.

If this is called with the messageNumber set to 0, email main assume the typeOrBoxName
argument is a mimetype and returns a new message structure. Message handling
components should call emh_getMessage with the number set to 0 and the mimetype
```

```
whenever a new message is started.  Normally this should be done whenever an active
message is trashed.
*/

em_MailData emh_getMessage (
        em_MessageNumber
        str       typeOrBoxName
)


/
**************************************************************
****/
/* emh_getRegisteredUsers
Returns a list of usernames for the users that are registered with the KidCode system, i.e.
that have been added as users by the User Adminstration part of the Main Email Program.
This is the same list of users that appear in the logon listbox when the program is started
up.  It may be used by installable components to create listboxes for filling address fields
in messages or for checking on whether a particular address is external to the system.
*/

em_RegisteredUsers   emh_getRegisteredUsers (
)


/
**************************************************************
****/
/* emh_sendMessage
Email Main sends the message argument (em_MailData) by either forwarding to an
external mail server or, if it is a registered KidCode user, writing the message to the user's
incoming mail mailbox.
*/

void emh_sendMessage (
        em_MailData
)


/
**************************************************************
****/
/* emh_saveMessage
Email Main saves the message argument (em_MailData) for the currently logged on user
by writing the message to the user's "notes in progress"  mail mailbox.
*/

void emh_saveMessage (
        em_MailData
)


/
**************************************************************
```

Appendix B: KidCode® API

```
****/
/* emh_disableButton
It is recommended that this function be used carefully. Normally Email Main controls
the state of all the buttons available to users to access message handling of the main
program (i.e. buttons in the purple left hand panel). This function can be used to request
that Email Main disable the button specified by the argument, ButtonName. If the button
is disabled - whether it was already disabled or is disabled as a result of the function call -
the function will return TRUE, otherwise it will return FALSE. The calling component
should check on whether the function call succeeded and proceed accordingly.
*/

em_ReturnValue emh_disableButton (
                str                 ButtonName
)


/
*****************************************************************
****/
/* emh_enableButton
It is recommended that this function be used carefully. Normally Email Main controls
the state of all the buttons available to users to access message handling of the main
program (i.e. buttons in the purple left hand panel). This function can be used to request
that Email Main enable the button specified by the argument, ButtonName. If the button
is enabled - whether it was already disabled or is disabled as a result of the function call -
the function will return TRUE, otherwise it will return FALSE. The calling component
should check on whether the function call succeeded and proceed accordingly.
*/

em_ReturnValue emh_enableButton (
                str                 ButtonName
)
```

API Functions Required Implementation of all Component Types

```
/
*****************************************************************
****/
/* emc_startMeUp
Used by Email Main to tell an installable component to start. This function will execute
prior to initialization of the component's data structures. Which should only be intialized
after the component receives the emc_initWindow call from Email Main.
This function is necessary for the Director/Lingo implementation.
*/

em_ReturnValue emc_startMeUp (
)


/
*****************************************************************
****/
/* emc_initWindow
Used by Email Main to tell an installable component to initialize it's data structures and
```

```
prepare its graphical display. The component is passed the username of the current user.
If it requires additional user information in order to initialize, it can call emh_getUserInfo
within it's implementation of this function.
*/

em_ReturnValue  emc_initWindow (
        em_UserName
)


/
*****************************************************************
****/
/* emc_closeWindow
Used by Email Main to tell an installable component to free all memory that it has used,
close it's window, and shut down.
*/

em_ReturnValue emc_closeWindow (
)


/
*****************************************************************
****/
/* emc_getComponentInfo
Used by Email Main to get required information such as componentName, componentID,
etc. from the installable component.
*/

em_ComponentInfo emc_getComponentInfo (
)
```

API Functions required of a Mailbox Handler Component

```
/
*****************************************************************
****/
/* mbx_getMessageNumbers
Used by Email Main to get the message number of the currently selected message in the
mailbox browser. If no message is selected, the list should be empty.
*/

list of int mbx_getMessageNumbers (
)


/
*****************************************************************
****/
/* mbx_getMessage
Used by Email Main to get the message data structure of the message with
em_MessageNumber from the mailbox currently displayed in the mailbox browser. If
the function fails, e.g. if there is no message with the given message number, the function
```

```
returns an empty list.
*/

em_MailData mbx_getMessage (
        em_MessageNumber
)


/
**************************************************************************
****/
/* mbx_trashMessages
Used by Email Main to tell the mailbox component to update it's display and it's data
structures to delete messages with messageNumbers in the argument list. If the function
fails, e.g. if one of the message numbers is invalid, the function returns FALSE, otherwise
it returns TRUE. This function should be implemented so that it does not perform partial
deletes, i.e. either it succeeds in deleting all of the messages in the list or it should not
delete any message.
*/

em_ReturnValue mbx_trashMessages (
        list of em_MessageNumber
)

/
**************************************************************************
****/
/* mbx_openMailbox
Used by Email Main to tell the mailbox component to display the mailbox passed in the
argument.
*/

em_ReturnValue mbx_openMailbox (
        em_Mailbox
)
```

Functions required of a Message Handler Component

```
/
**************************************************************************
****/
/* msh_sendMessage
Used by Email Main to tell a message handling component to pass back a fully completed
message data structure so that it can be sent to the recipient specified in the message's
address field. The message handling component should update it's display as appropriate
for a message that has been sent. It should also change it's state to #display mode because
a message that has already been sent should not be editable. If the function fails, e.g. if a
fully completed message cannot be constructed (for example, if the user has not specified
a message recipient), the function returns an empty list.

The message handling component will normally control all dialogs with a user that pertain
to the message under construction. For example to alert the user to the fact that a
message recipient is required. However, if the message handling component fails to pass
back a properly formatted, completed message data structure, (or an empty list
```

acknowledging failure) Email Main will detect the error and alert the user about the field or fields that have not been specified.
*/

```
em_MailData  msh_sendMessage (
)
```

/
****
****/
/* msh_openMessage
Used by Email Main to pass a message data structure to a message handling component so that it can be displayed. The message handling component should display the message in the specified mode - either #author or #display. If the em_Mode argument is #display the message should not be editable. Otherwise the message should be opened so that it can be edited.

If the function fails, e.g. if an error is detected in the message body, the message handler returns FALSE, otherwise the message handler returns TRUE.
*/

```
em_ReturnValue  msh_openMessage (
        em_MailData
        em_Mode
)
```

/
****
****/
/* msh_replyMessage
Used by Email Main to inform a message handling component to display the currently active message for editing as a reply. In order to reply the message handing component will generally create a new message with the mode set to #author. The new message body may contain material from the original message that is being replied to. In addition, message handling components that handle different player roles may enable or disable various role specific tools at this time. For example, the Rebus message handler will change the RebusState of the new message and enable guessboxes as appropriate.

If the function fails, e.g. if an error is detected in the message body, the message handler returns FALSE, otherwise the message handler returns TRUE.
*/

```
em_ReturnValue  msh_replyMessage (
)
```

/
****
****/
/* msh_clearMessage
Used by Email Main to inform a message handling component that the current message should be cleared from the display and from the message handling component's data structures. This function is used, for example, when the user indicates they want to trash the current message by clicking on the "trash" button in the Email Main purple panel.

If the function fails, the message handler returns FALSE. Otherwise the message handler returns TRUE.
*/

```
em_ReturnValue  msh_clearMessage (
)


/
*****************************************************************
****/
/* msh_printMessage
```

Used by Email Main to inform a message handling component that a message should be printed. This function is used, for example, when the user indicates they want to print the current message by clicking on the "print" button in the Email Main purple panel. When the argument, em_mailData, is an empty list, the message handler component should print the currently active message. Otherwise the message handler component should print the message argument. Normally, if the message handler component has been fully initialized and is displayed in a window, Email Main will call this function with an empty list for an argument.

The function may also be used by the Main Email program to have a message handler print a message even though the message handler component has not been fully initialized and displayed in a window. For example, this will occur if an active mailbox component receives a print request from Email Main for a message that has been selected in the mailbox browser. In this case, Email Main will send a request to the appropriate message handler component to print the message without fully starting it up and initializing its window. Therefore the message handler should implement the msh_printMessage function so that the following sequence of function calls succeeds - emc_startMeUp, msh_printMessage(message).

If the function fails, the message handler returns FALSE. Otherwise the message handler returns TRUE.
*/

```
em ReturnValue  msh_printMessage (
        em_MailData
)
```

Claims:

1. An electronic mail client, comprising:
a) a plurality of authoring components, a first of said plurality of authoring components for creating a representation of a document including other than text;
b) encoding means for automatically encoding said representations created with said authoring components into an Internet-compatible email message; and
c) decoding means for automatically decoding said representations encoded by said encoding means.

2. An electronic mail client according to claim 1, wherein:
said plurality of authoring components include at least one installable component.

3. An electronic mail client according to claim 1, wherein:
said plurality of authoring components includes at least one component selected from the group consisting of a game component, a spreadsheet component, and a graphic editor component.

4. An electronic mail client according to claim 1, wherein:
said plurality of authoring components includes at least one component selected from the group consisting of a database component, a presentation component, and a puzzle component.

5. An electronic mail client according to claim 1, wherein:
said encoding means includes MIME-compatible encoding means.

6. An electronic mail client according to claim 1, wherein:
said encoding means includes means for creating a MIME file and means for creating a multipart MIME message,
each of said authoring component cooperating with said encoding means such that a creation of said MIME file and said multipart MIME message is transparent to a user.

7. An electronic mail client according to claim 6, wherein:
said decoding means includes means for concatenating a multipart MIME message and means for decoding a MIME file;
each of said authoring component cooperating with said decoding means such that a concatenation of said multipart MIME message and said decoding of MIME files is transparent to the user.

8. An electronic mail client according to claim 1, further comprising:
d) a plurality of installable mailbox/browser components, each of said mailbox/browser components displaying different types of documents in a user's mailbox.

9. An electronic mail client according to claim 1, further comprising:
d) a plurality of installable mailbox/browser components, each of said mailbox/browser components displaying mailbox contents in a different style.

10. An electronic mail client according to claim 1, wherein:
said encoding means and said decoding means communicate bidirectionally with said authoring components.

11. An electronic mail client according to claim 1, wherein:
at least one of said authoring components includes means for recognizing whether a user is an author or a reader and for responding differently to authors and readers.

12. An electronic mail client according to claim 1, wherein:
at least one of said authoring components includes means for allowing a user to create a read-only document.

13. An electronic mail client for a student and a teacher, comprising:
a) a plurality of authoring components, a first of said plurality of authoring components for creating a representation of a text document and a second of said plurality of authoring components for creating a representation of a document including other than text;
b) encoding means for automatically encoding representations created with said authoring components into an email message; and
c) decoding means for automatically decoding said representations encoded with said encoding means, wherein
at least one of said authoring components includes means for determining whether the user is the student or the teacher.

14. An electronic mail client according to claim 13, wherein:
said plurality of authoring components include at least one installable component.

15. An electronic mail client according to claim 13, wherein:

said plurality of authoring components includes at least one component selected from the group consisting of a game component, a workbook component, and a graphic editor component.

16. An electronic mail client according to claim 13, wherein:

said plurality of authoring components includes at least one component selected from the group consisting of a database component, a presentation component, and a puzzle component.

17. An electronic mail client according to claim 13, wherein:

said encoding means includes MIME-compatible encoding means.

18. An electronic mail client according to claim 13, wherein:

said encoding means includes means for creating a MIME file and means for creating a multipart MIME message,

each of said authoring components cooperating with said encoding means such that a creation of said MIME file and said multipart MIME message is transparent to the student and the teacher.

19. An electronic mail client according to claim 18, wherein:

said decoding means includes means for concatenating a multipart MIME message and means for decoding a MIME file,

each of said authoring component cooperating with said decoding means such that a concatenation of said multipart MIME message and said decoding of MIME files is transparent to a user.

20. A method of authoring a document and sending it by electronic mail, said method comprising:

a) providing a document-authoring component which authors a document other than a plain-text document;

b) providing a document-encoding component which encodes the document as Internet-compatible email;

c) linking the document-authoring component with the document-encoding component such that documents generated under said document-authoring component are automatically encoded as Internet-compatible email.

21. A method according to claim 20, wherein:

said step of providing a document-authoring component includes providing a plurality of document-authoring components, and

said step of linking includes linking each of said document-authoring components with the document-encoding component.

22. A method according to claim 20, further comprising:

d) providing a document-decoding component which decodes a received document encoded as Internet-compatible email;

c) linking the document-authoring component with the document-decoding component such that documents are automatically decoded.

23. A method according to claim 20, wherein:

the document-encoding component includes means for creating a MIME file and means for creating a multipart MIME message.

24. A method according to claim 22, wherein:

the document-decoding component includes means for concatenating a multipart MIME message and means for decoding a MIME file.

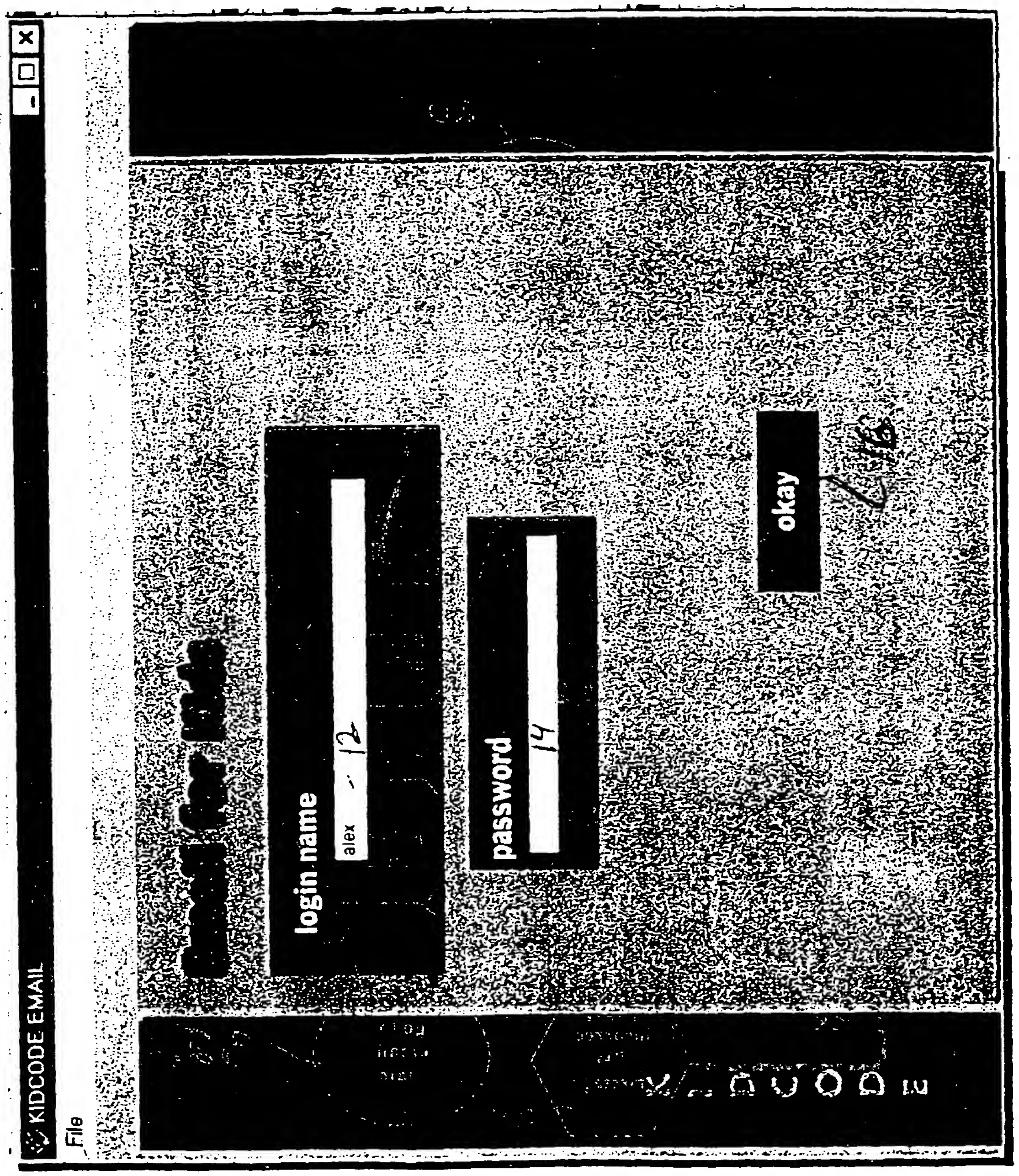


Fig. 1

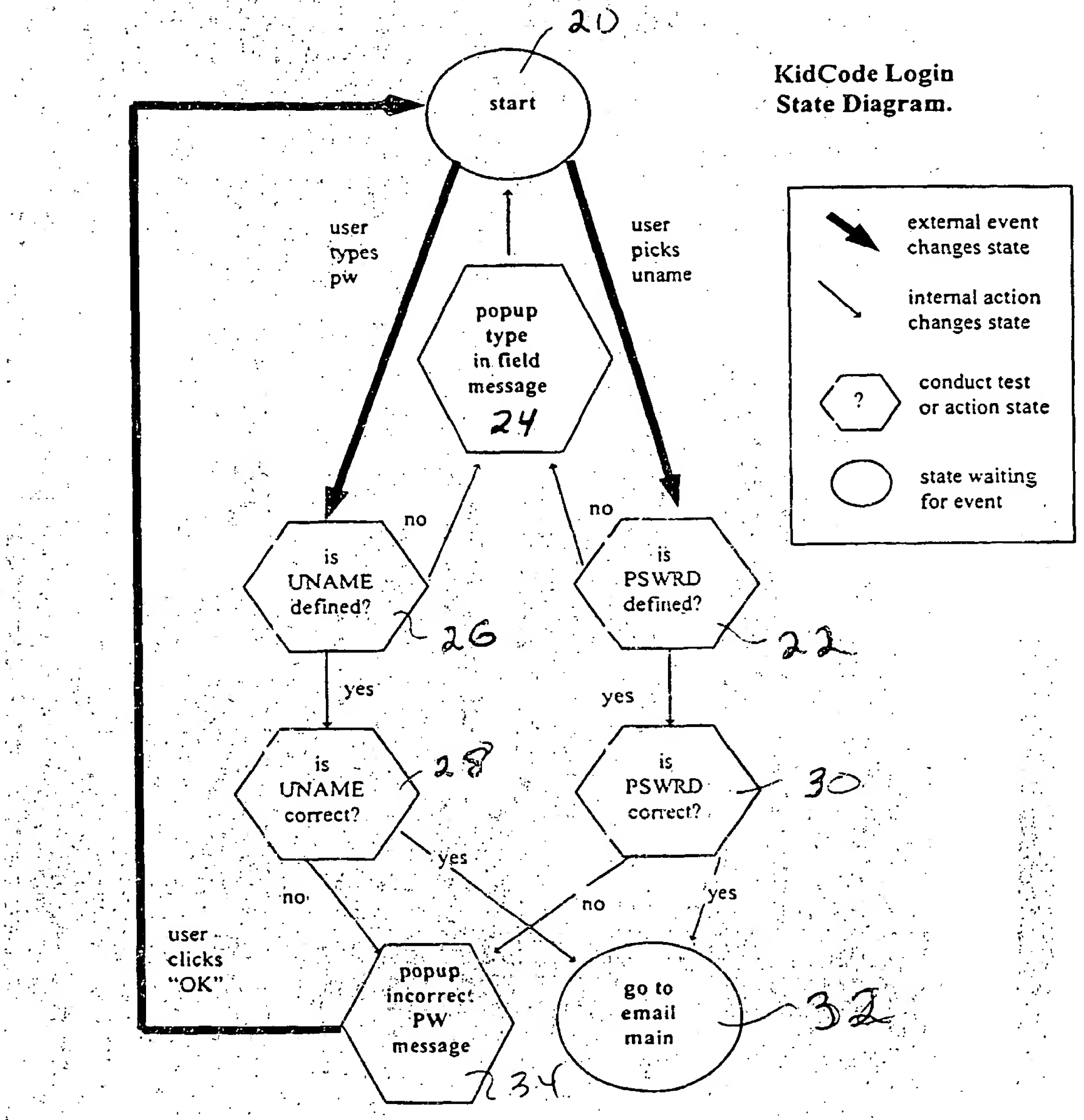

KidCode Login
State Diagram.
20
start
user types pw
user picks uname
popup type in field message
24
no
is UNAME defined?
26
is PSWRD defined?
22
yes
is UNAME correct?
28
is PSWRD correct?
30
yes
no
user clicks "OK"
popup incorrect PW message
34
go to email main
32
external event changes state
internal action changes state
?
conduct test or action state
state waiting for event

Fig. 1a

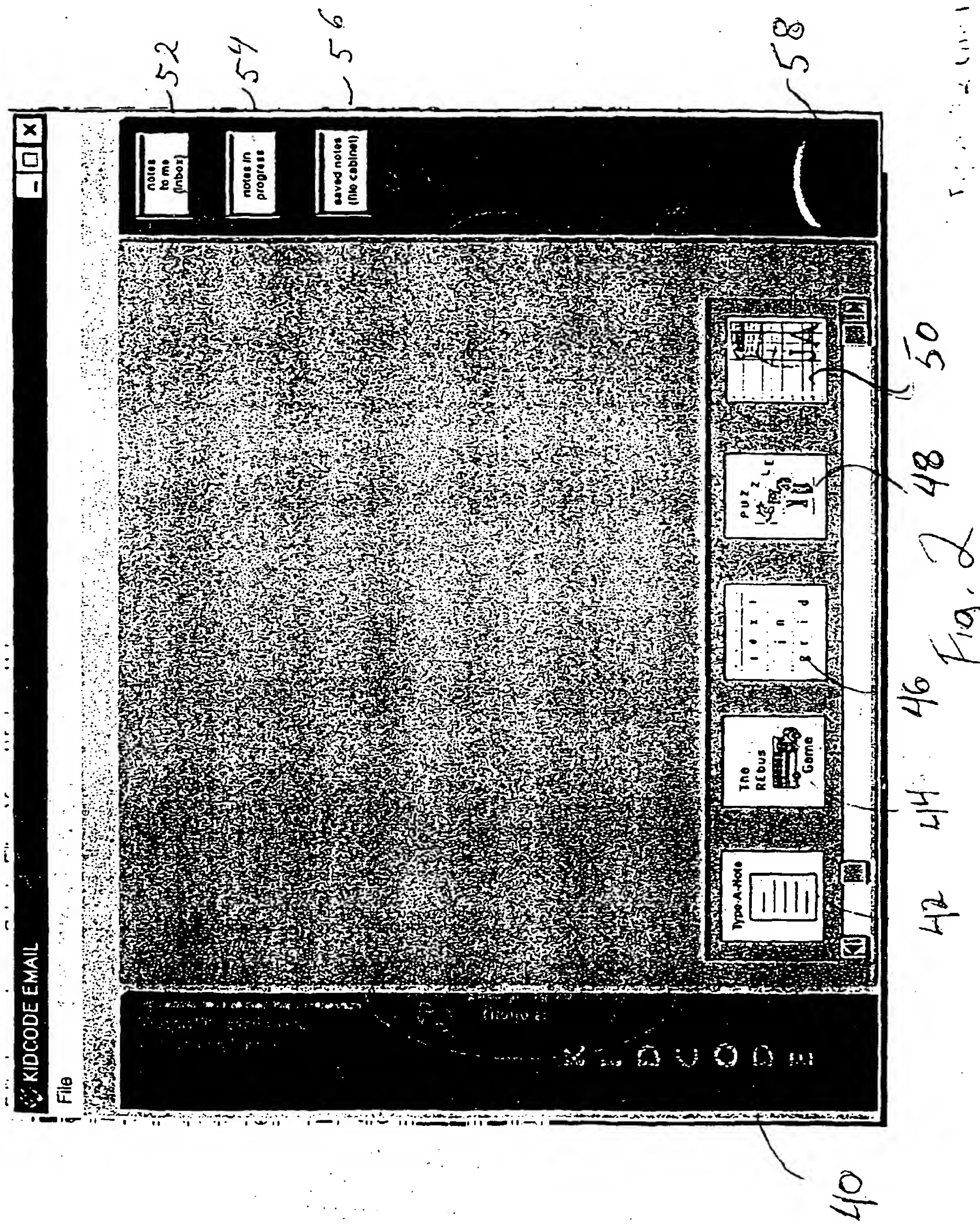

KIDCODE EMAIL
File
notes to me (inbox)
notes in progress
saved notes (file cabinet)
Type-A-Note
The REbus Game
PUZZLE
52
54
56
58
40
42
44
46
48
50

Fig. 2

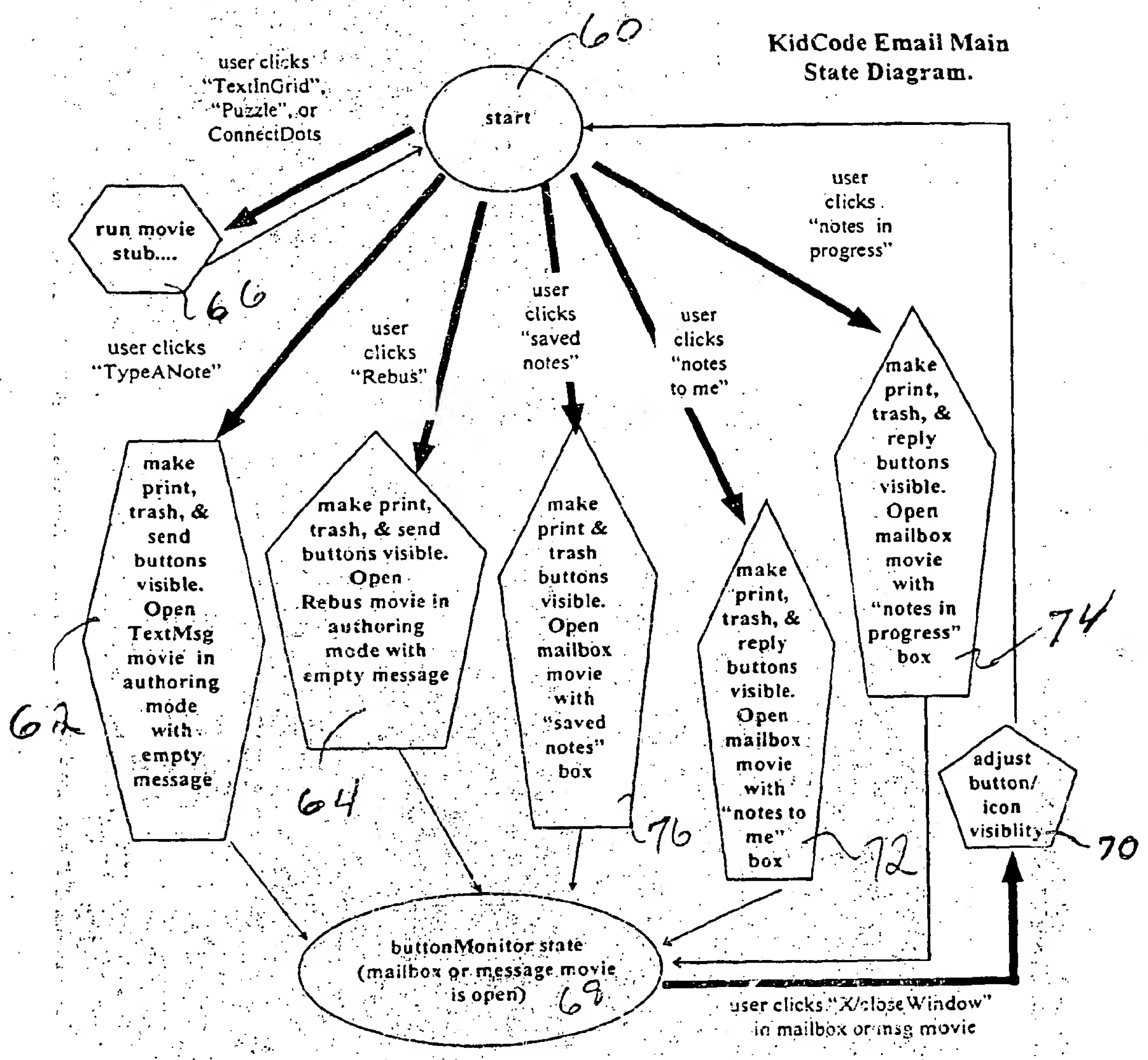
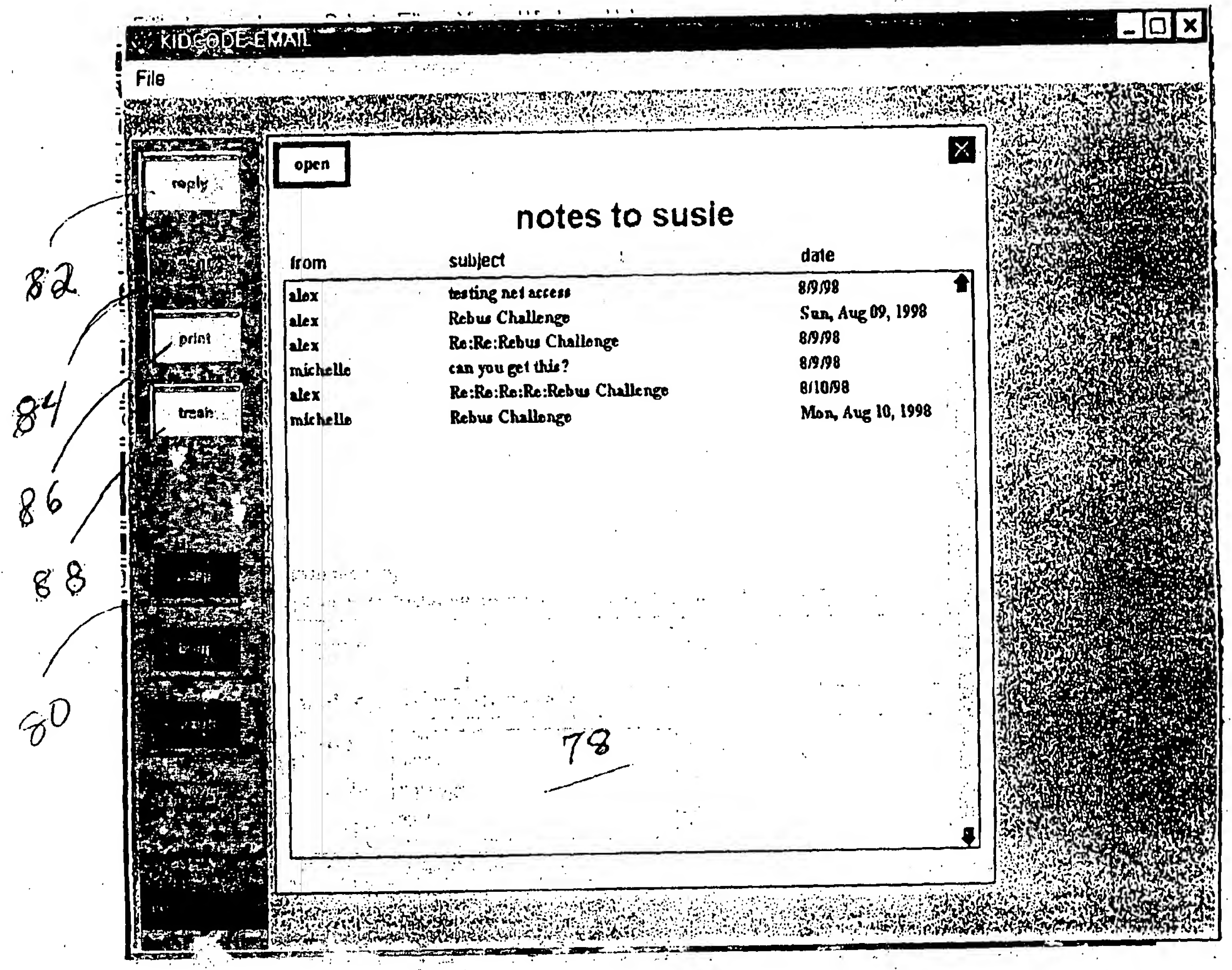

KidCode Email Main State Diagram.
start
60
user clicks "TextInGrid", "Puzzle", or ConnectDots
run movie stub....
66
user clicks "TypeANote"
user clicks "Rebus"
user clicks "saved notes"
user clicks "notes to me"
user clicks "notes in progress"
make print, trash, & send buttons visible. Open TextMsg movie in authoring mode with empty message
62
make print, trash, & send buttons visible. Open Rebus movie in authoring mode with empty message
64
make print & trash buttons visible. Open mailbox movie with "saved notes" box
76
make print, trash, & reply buttons visible. Open mailbox movie with "notes to me" box
72
make print, trash, & reply buttons visible. Open mailbox movie with "notes in progress" box
74
adjust button/ icon visiblity
70
buttonMonitor state (mailbox or message movie is open)
68
user clicks "X/closeWindow" in mailbox or msg movie

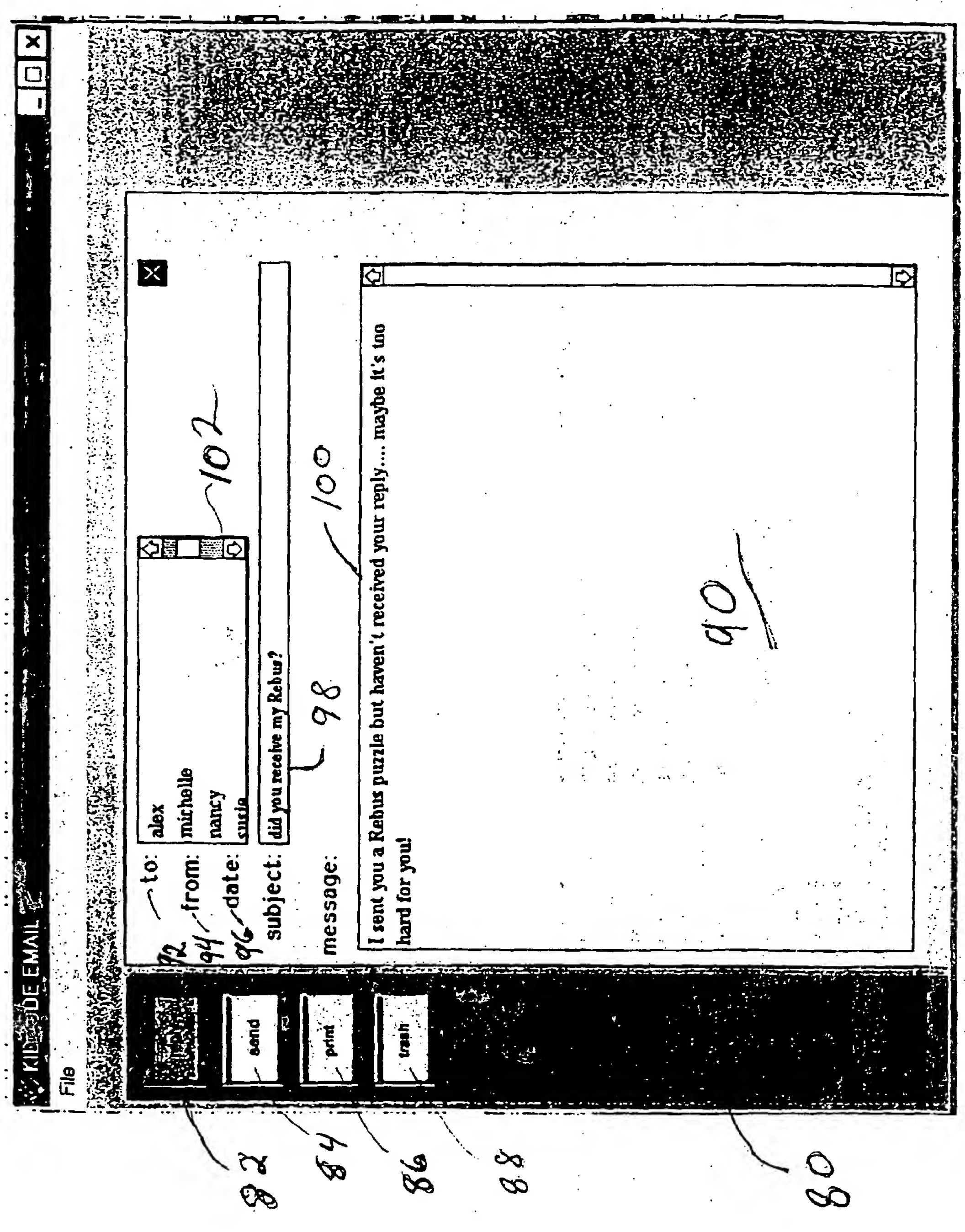
Fig. 2a



notes to susie

from	subject	date
alex	testing net access	8/9/98
alex	Rebus Challenge	Sun, Aug 09, 1998
alex	Re:Re:Rebus Challenge	8/9/98
michelle	can you get this?	8/9/98
alex	Re:Re:Re:Re:Rebus Challenge	8/10/98
michelle	Rebus Challenge	Mon, Aug 10, 1998

Fig. 3



did you receive my Rebus?

I sent you a Rebus puzzle but haven't received your reply.... maybe it's too hard for you!

Fig. 4

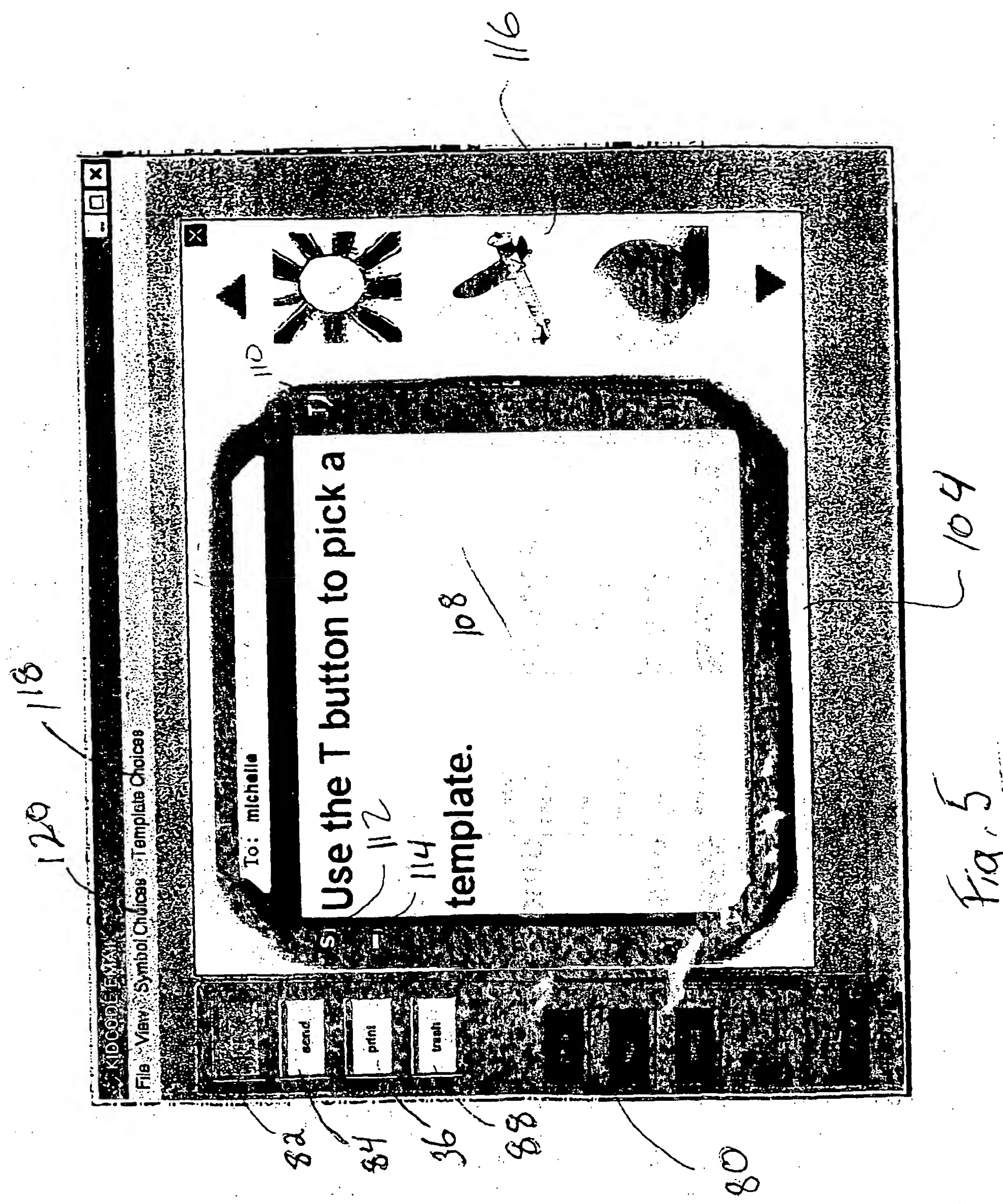

KIDCODE EMAIL
File View Symbol Choices Template Choices
To: michelle
Use the T button to pick a template.
S
T
send
print
trash
80
82
84
86
88
104
108
110
112
114
116
118
120

Fig. 5

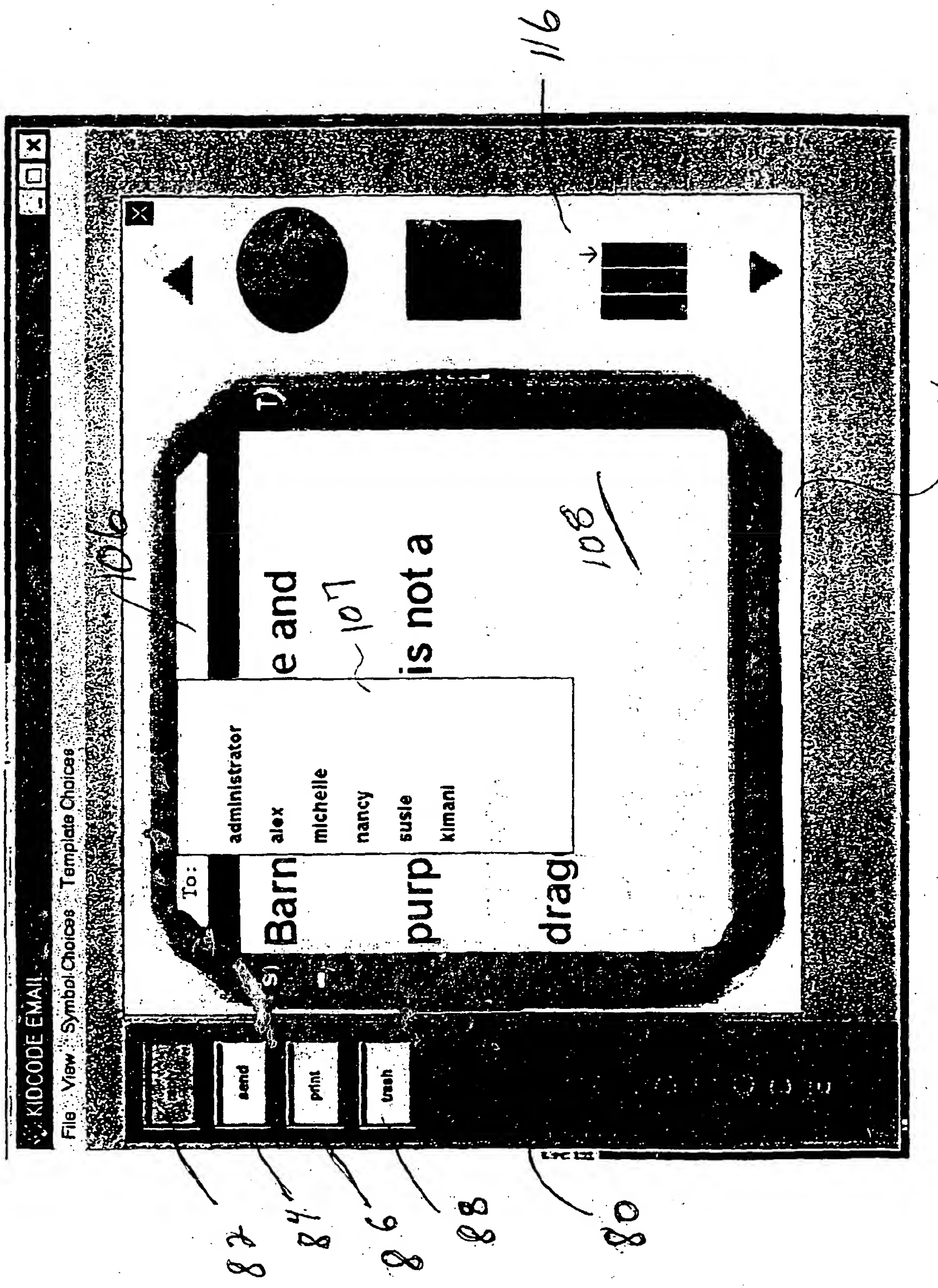
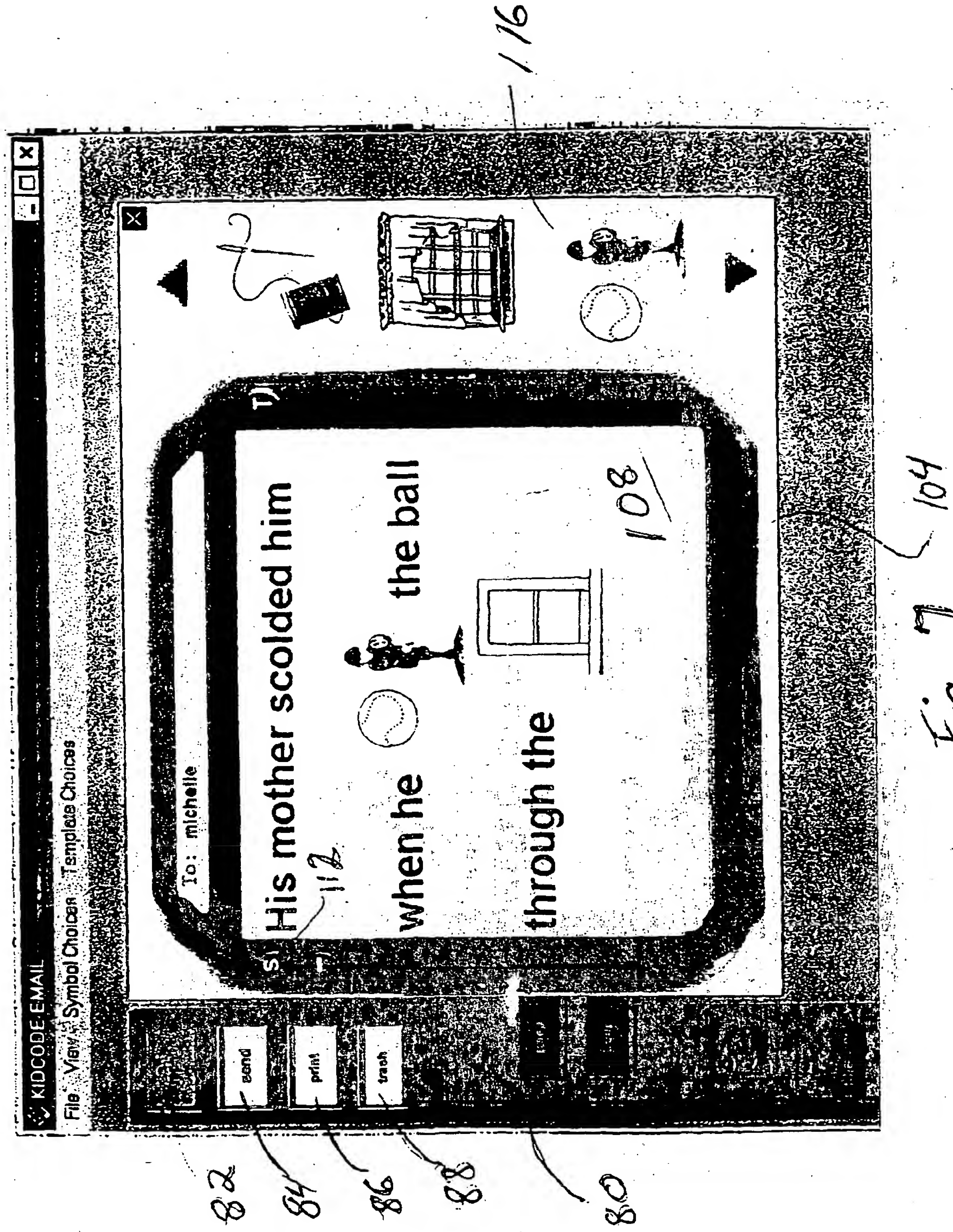

KIDCODE EMAIL
File View Symbol Choices Template Choices
To:
administrator
alex
michelle
nancy
susie
kimani
Barn e and
purp is not a
drag
send
print
trash
116
104
106
107
108
82
84
86
88
80

Fig. 6



His mother scolded him

when he the ball

through the

Fig. 7

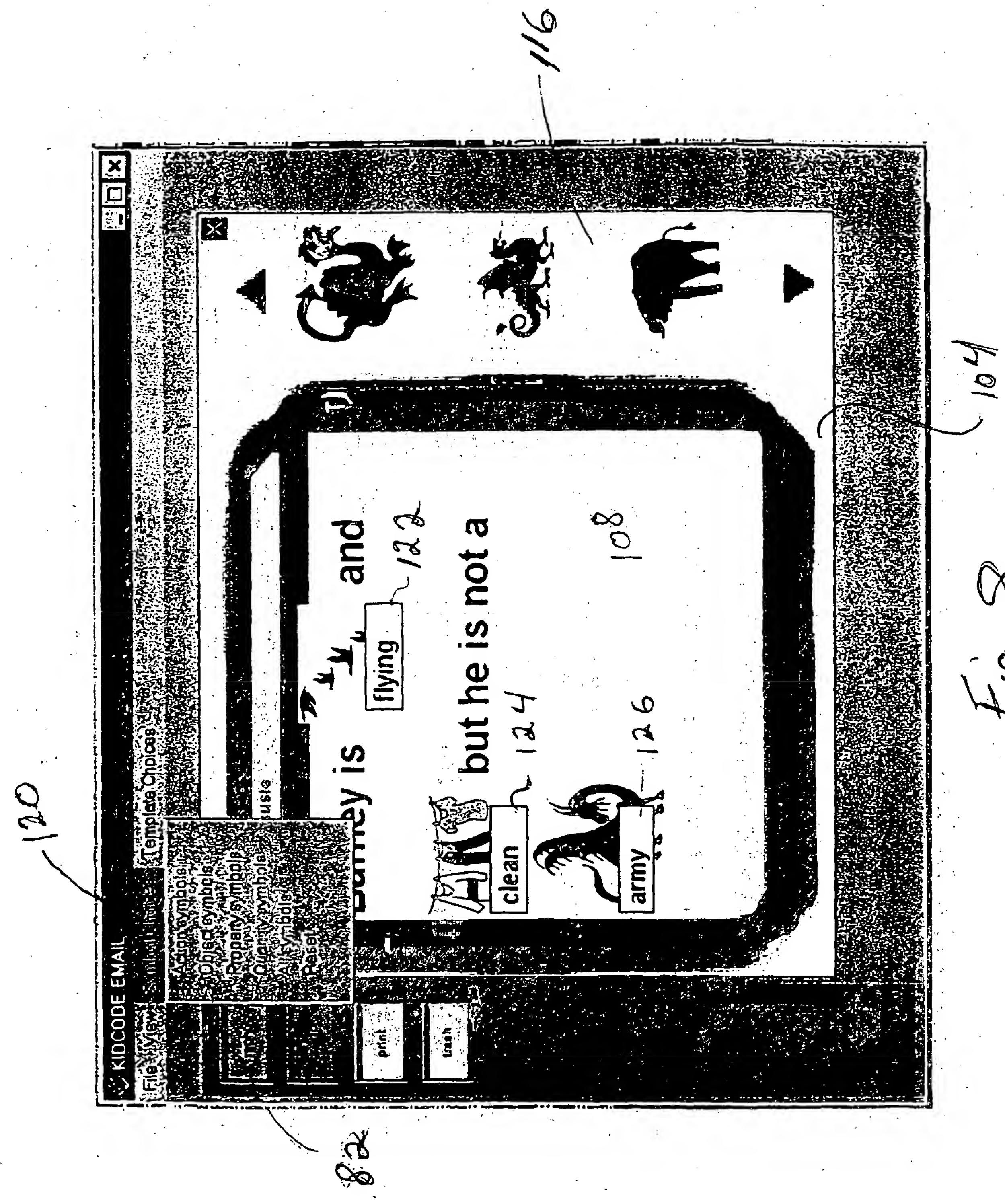
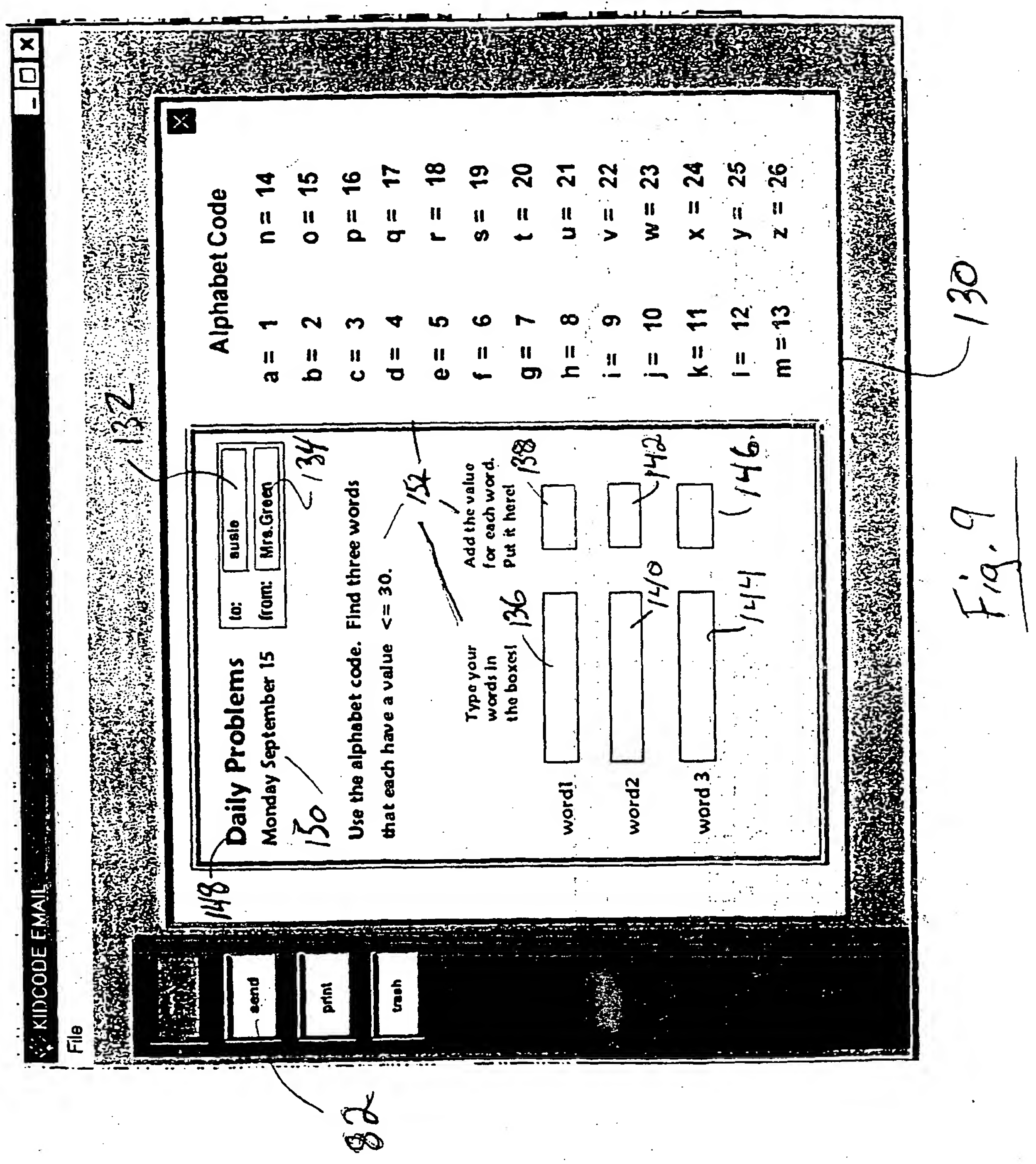

KIDCODE EMAIL
File
View
Template Choices
Action symbols
Object symbols
Property symbols
Quantity symbols
All Symbols
Reset
ey is
flying
and
but he is not a
clean
army
print
trash
116
104
122
108
124
126
120
82

Fig. 8



Daily Problems

Monday September 15

to: susie

from: Mrs.Green

Use the alphabet code. Find three words that each have a value <= 30.

Type your words in the boxes!

Add the value for each word. Put it here!

word1

word2

word 3

Alphabet Code

a = 1	n = 14
b = 2	o = 15
c = 3	p = 16
d = 4	q = 17
e = 5	r = 18
f = 6	s = 19
g = 7	t = 20
h = 8	u = 21
i = 9	v = 22
j = 10	w = 23
k = 11	x = 24
l = 12	y = 25
m = 13	z = 26

Fig. 9

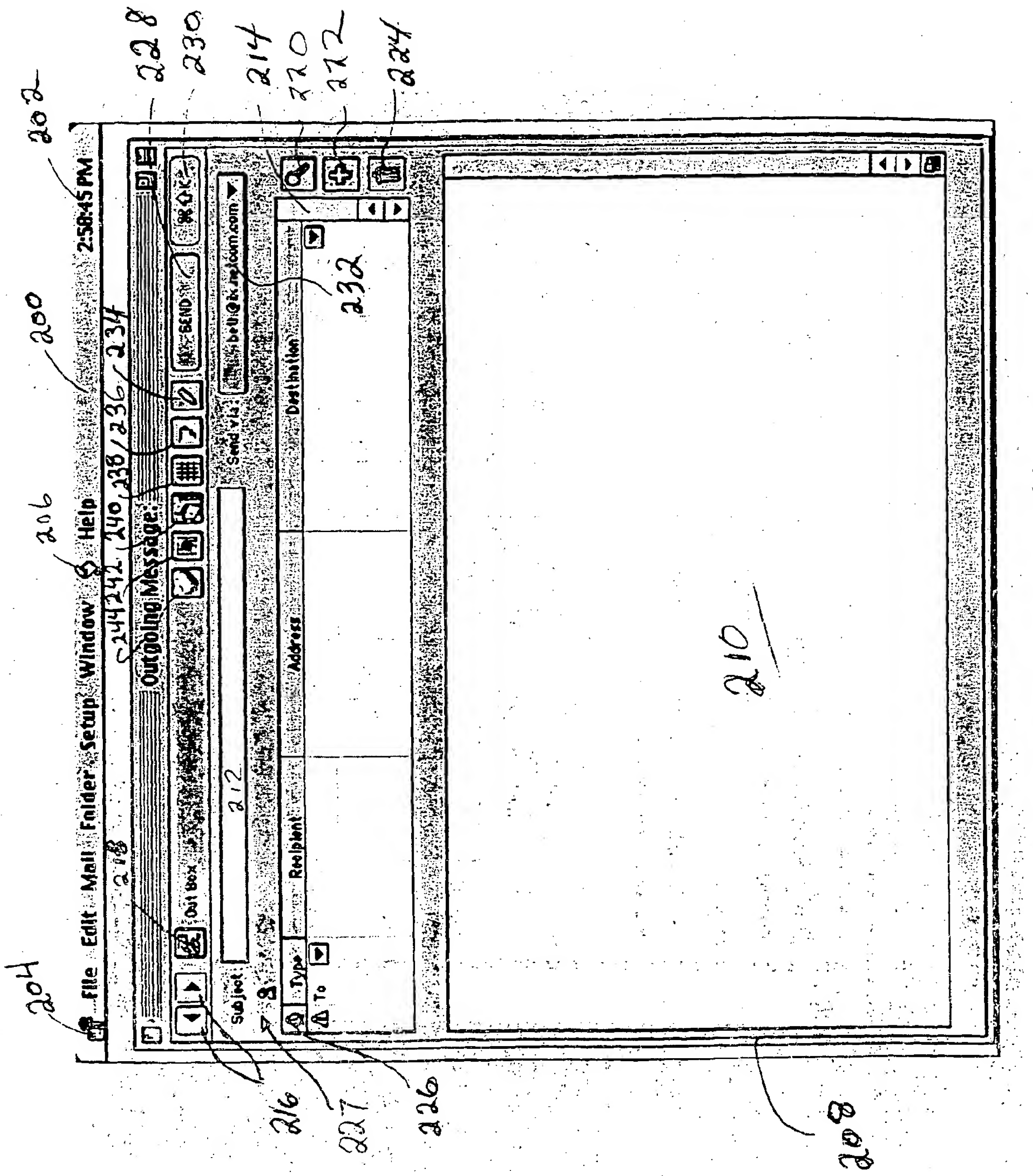

File Edit Mail Folder Setup Window Help
2:58:45 PM
Outgoing Message:
Out Box
SEND
Subject
Send via:
beth@kinetcom.com
Type
Recipient
Address
Destination
To
200
202
204
206
208
210
212
214
216
218
220
222
224
226
227
228
230
232
234
236
238
240
242
244

Fig. 10

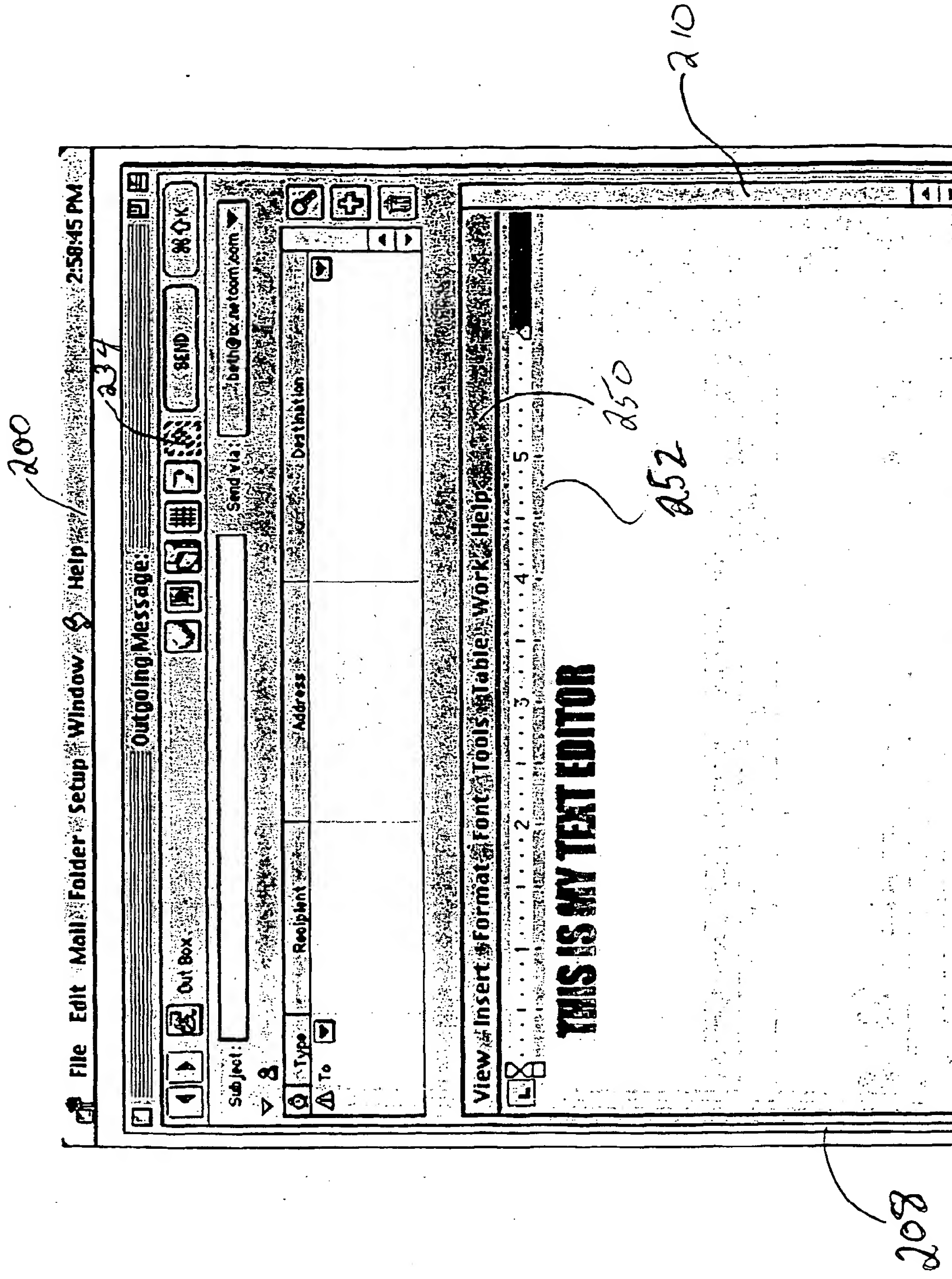

File
Edit
Mail
Folder
Setup
Window
Help
2:58:45 PM
Outgoing Message
Out Box
SEND
Subject:
Send via:
beth@ix.netcom.com
Type
Recipient
Address
Destination
To
View
Insert
Format
Font
Tools
Table
Work
Help
THIS IS MY TEXT EDITOR
200
234
250
252
210
208

Fig. 11

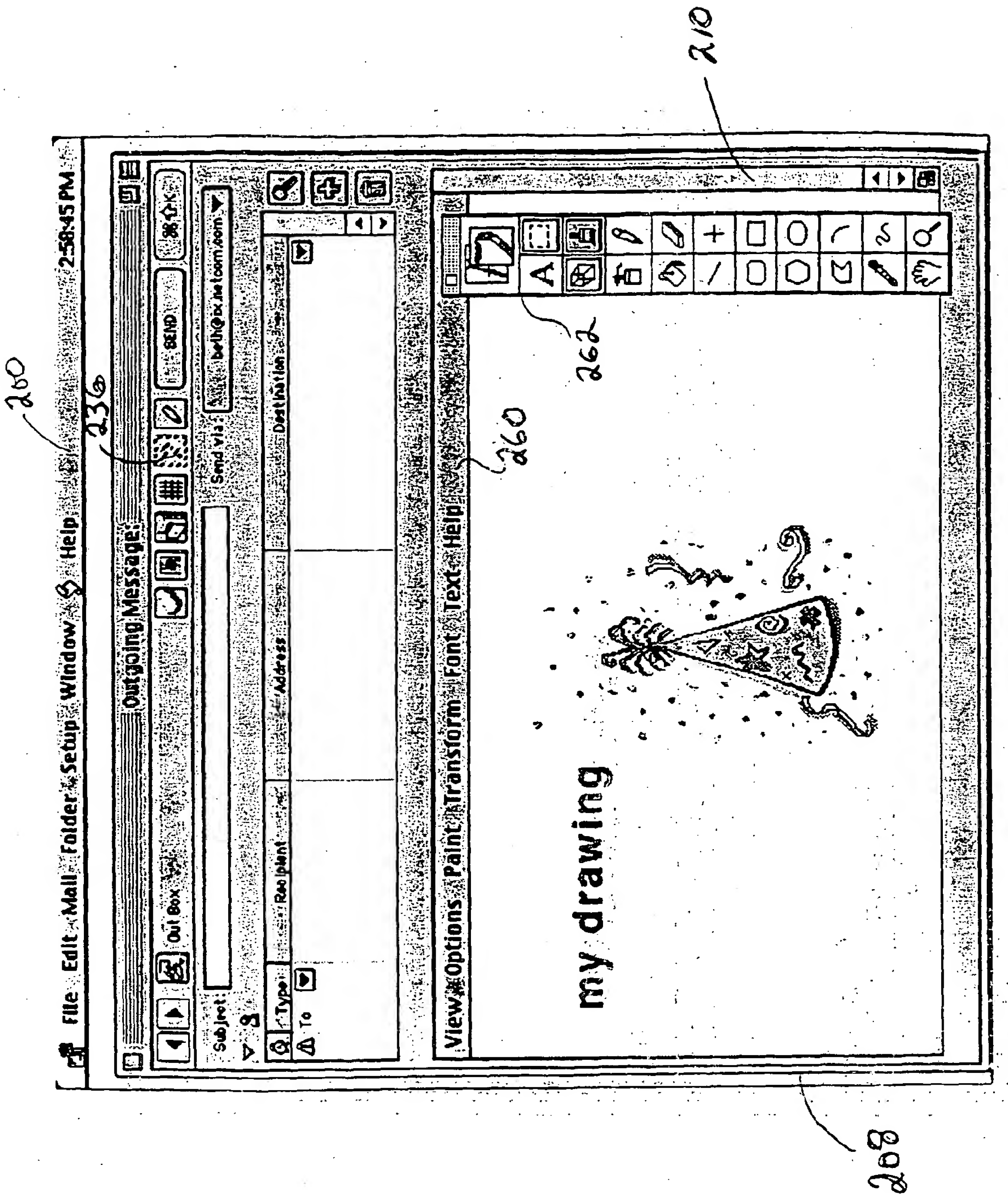

File Edit Mail Folder Setup Window Help 2:58:45 PM
Outgoing Message
Out Box
SEND
Subject:
Send via:
beth@ix.netcom.com
Type
Recipient
Address
Destination
To
View Options Paint Transform Font Text Help
my drawing
200
236
260
262
210
208

Fig. 12

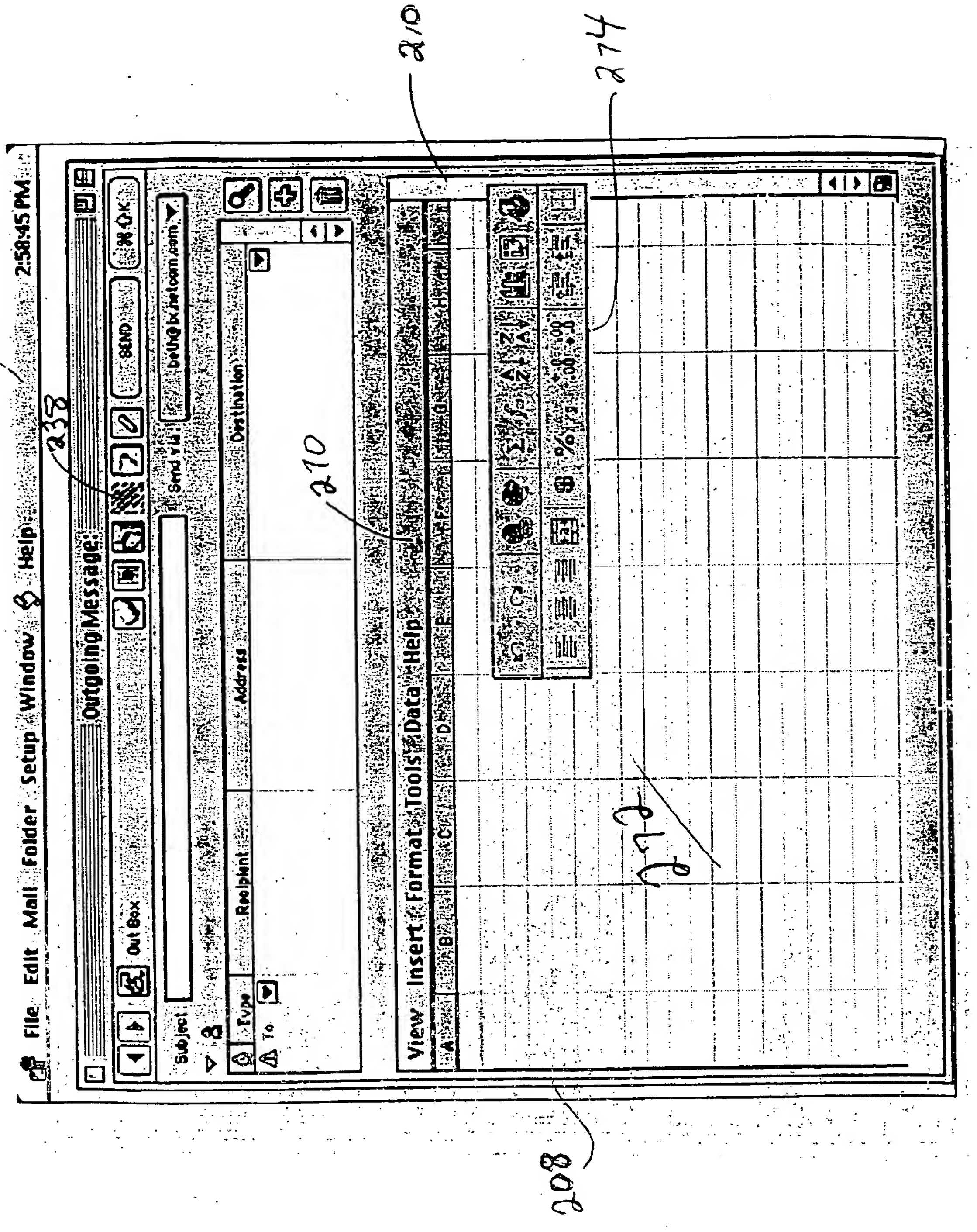

200
File Edit Mail Folder Setup Window Help 2:58:45 PM
Outgoing Message:
Out Box
SEND
Subject:
Send via:
Type
Recipient
Address
Destination
To
238
210
270
274
View Insert Format Tools Data Help
A B C D E F G H I
272
208

Fig. 13

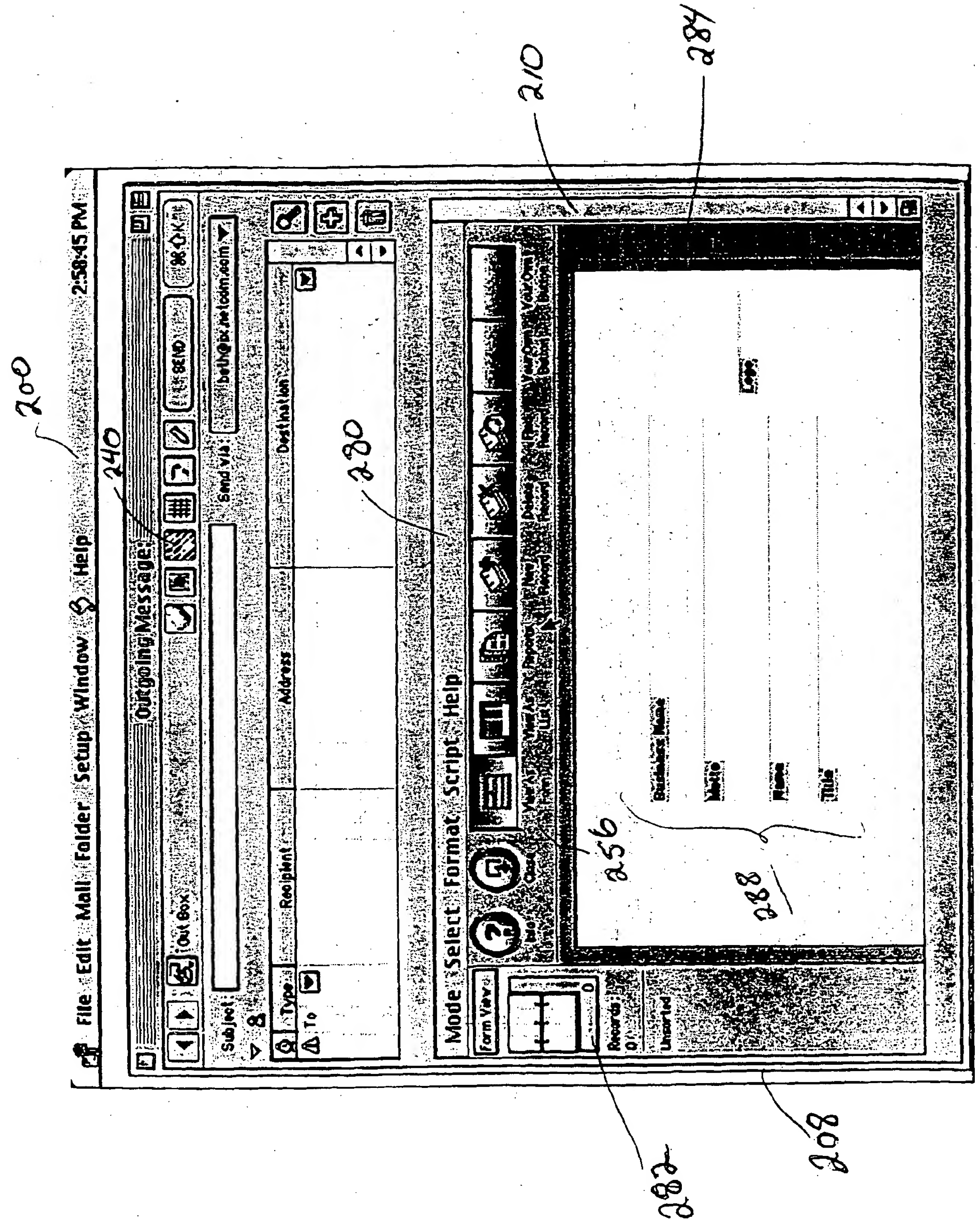

200
240
210
284
280
256
288
282
208
File Edit Mail Folder Setup Window Help
2:58:45 PM
Outgoing Message:
Out Box
SEND
Subject
Send via:
beth@pic.netcom.com
Type
Recipient
Address
Destination
To
Mode Select Format Script Help
Form View
Records:
Unsorted
Business Name
Motto
Name
Title
Logo

Fig. 14

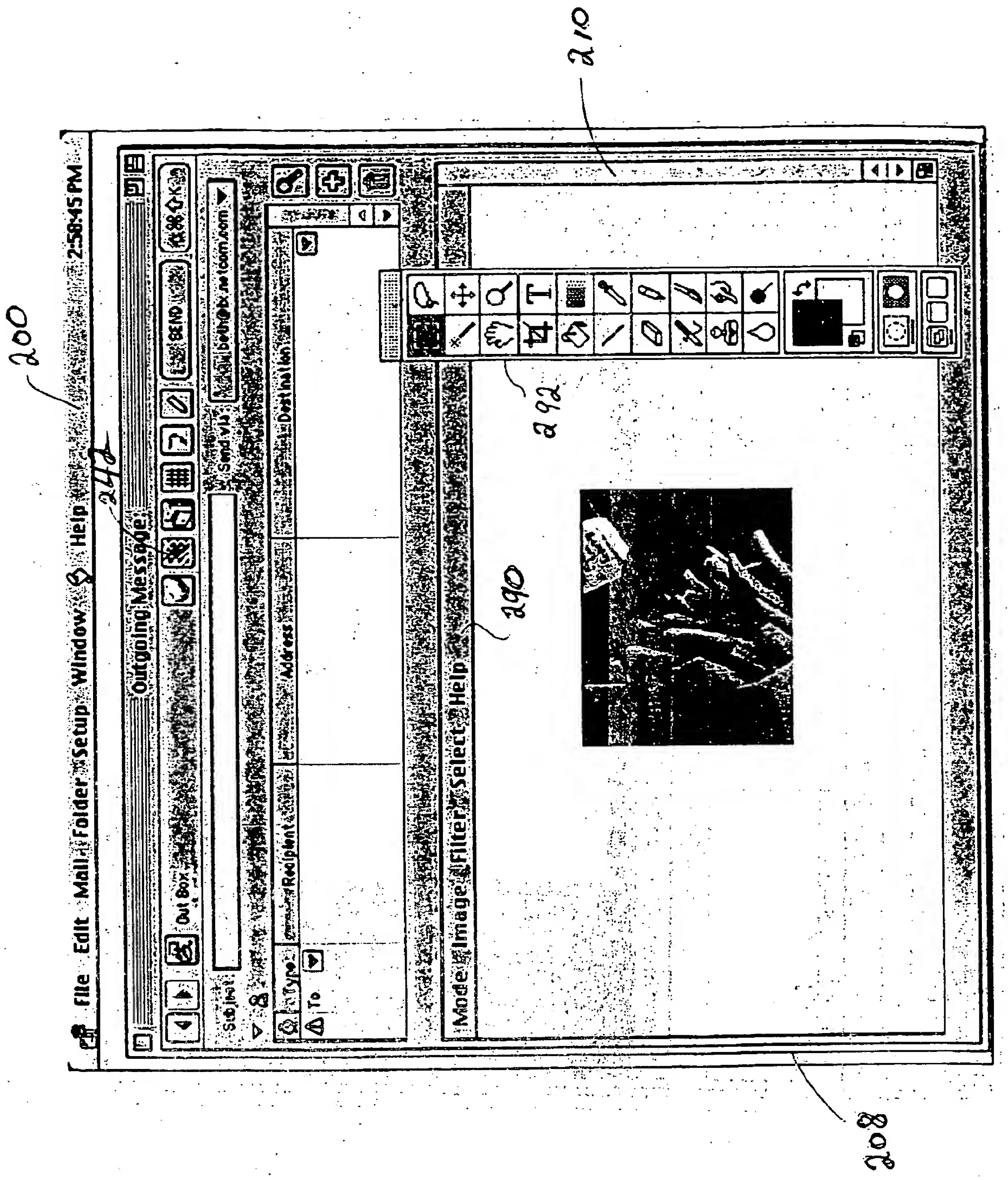
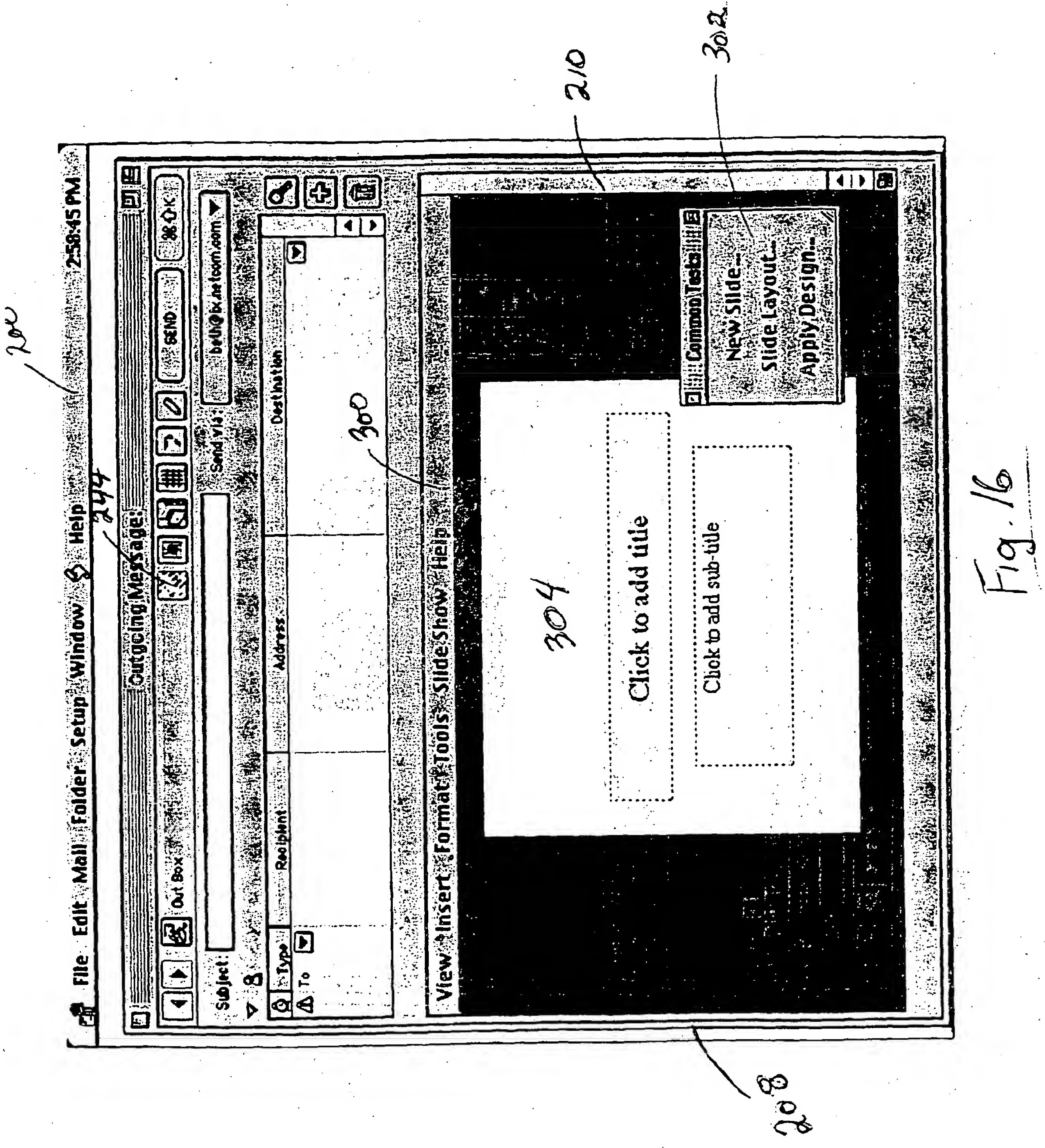
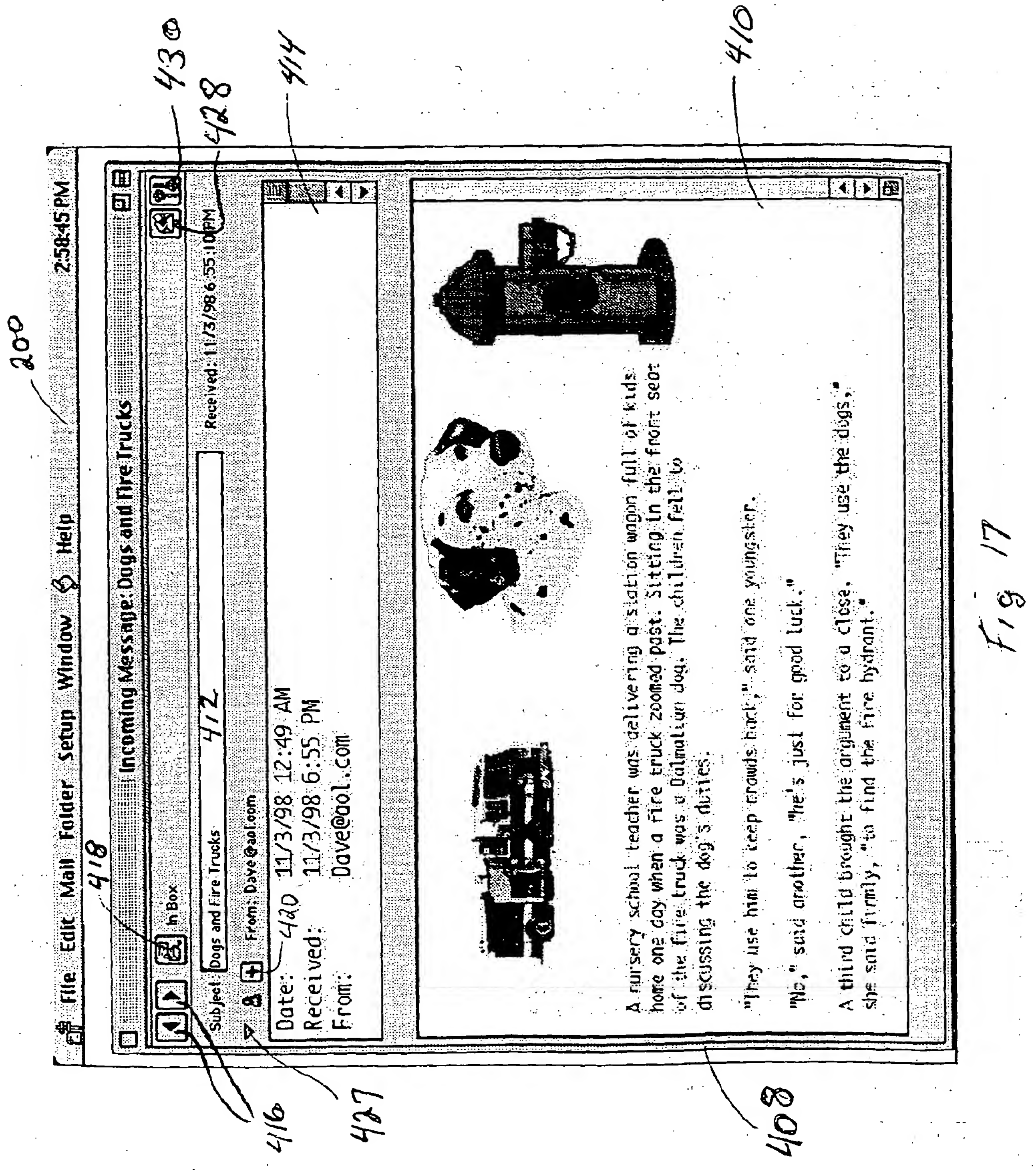

File
Edit
Mail
Folder
Setup
Window
Help
2:58:45 PM
Outgoing Message
Out Box
SEND
Subject:
Send via
beth@bx.netcom.com
Type
Recipient
Address
Destination
To
Mode
Image
Filter
Select
Help
200
242
210
290
292
208

Fig. 15



200
File Edit Mail Folder Setup Window Help
2:58:45 PM
Outgoing Message
244
Out Box
SEND
Subject:
Send via: beth@bc.netcom.com
Type
Recipient
Address
Destination
To
300
View Insert Format Tools SlideShow Help
210
302
Common Tasks
New Slide...
Slide Layout...
Apply Design...
304
Click to add title
Click to add sub-title
208

Fig. 16


200
430
428
414
410
418
412
420
416
427
408
File Edit Mail Folder Setup Window Help
2:58:45 PM
Incoming Message: Dogs and Fire Trucks
In Box

Subject: Dogs and Fire Trucks

Received: 11/3/98 6:55:10 PM

From: Dave@aol.com

Date: 11/3/98 12:49 AM
Received: 11/3/98 6:55 PM
From: Dave@aol.com

A nursery school teacher was delivering a station wagon full of kids home one day when a fire truck zoomed past. Sitting in the front seat of the fire truck was a Dalmatian dog. The children fell to discussing the dog's duties.

"They use him to keep crowds back," said one youngster.

"No," said another, "he's just for good luck."

A third child brought the argument to a close. "They use the dogs," she said firmly, "to find the fire hydrant."

Fig 17

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/29368

A. CLASSIFICATION OF SUBJECT MATTER
IPC(6) : G06F 3/14
US CL : 707/526, 530, 531, 709/206
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 707/526, 530, 531, 709/206

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EAST, WEST, ITKNOWLEGE.COM

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A, Y	US 5,710,883 A (HONG et al) 20 January 1998, column 2-50	5-7, 17-19, 23-24
A, Y	US 5,818,447 A (WOLF et al) 06 October 1998, column 23	3-4, 8-9, 12, 15-16, 21-22
X --- Y	FLEMING, H. Internet Explorer 4 6-in-1, Macmillan Comp. Pub., pages. 182-185, 201-205, 210-217, 265-271	1-2, 10-11, 14,-20 --- 3-9, 12-13, 21-24
Y	GUILDFORD, E. Netscape Communicator 6-in-1, Macmillan Comp. Pub., pages 218-223, 259-262, 264-268	3-4, 8-9, 12, 15-16, 21-22
A, E	US A 6,014,688 A (VENKATRAMAN et al) 11 January 2000, column 1, lines 50-67, column 2, lines 1-67.	3-4, 8-9, 12, 15-16, 21-22
A, E	US A 5,995,756 A (HERRMAN et al) 30 November 1999, column 3, lines 20-67.	3-4, 8-9, 12, 15-16, 21-22
A, Y	US A 5,706,434 A (KREMEN et al) 06 January 1998, column 3, lines 1-67.	3-4, 8-9, 12, 15-16, 21-22
A, P	US A 5,889,518 A (POREH et al) 30 March 1999, column73, lines 20-67.	3-4, 8-9, 12, 15-16, 21-22
A, Y	US A 5,471,470 A (SHARMA et al) 28 November 1995, column 2, lines 20-67.	3-4, 8-9, 12, 15-16, 21-22

[✓] Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report 06 APR 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703)305-3230	Authorized officer STEPHEN HONG James R. Matthews Telephone No. (703) 305-3900

INTERNATIONAL SEARCH REPORT

Inte onal application No.
PCT/US99/29368

C (Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A, Y	US A 5,835,769 A (JERVIS et al) 10 November 1998, column 2, lines 1-56.	3-4, 8-9, 12, 15-16, 21-22
A, Y	US A 5,452,289 A (SHARMA et al) 19 September 1995, column 2, lines 8-67.	3-4, 8-9, 12, 15-16, 21-22